CUNY SPS DATA 621 - CTG5 - Final

Betsy Rosalen, Gabrielle Bartomeo, Jeremy O'Brien, Lidiia Tronina, Rose Jones

May 25th, 2019

Contents

1 PROJECT DESCRIPTION AND BACKGROUND

1.1 Background

With nearly 18MM deaths in 2015, cardiovascular diseases (CVD) are the leading cause of death globally and growing in the developing world. CVD is a disease class which includes heart attacks, strokes, heart failure, coronary artery disease, arrhythmia, venous thrombosis, and other conditions. About half of all Americans (47%) have at least one of three key risk factors for heart disease: high blood pressure, high cholesterol, and smoking.

Researchers estimate that up to 90% of heart disease deaths could be prevented. Typical means of detection include electrocardiograms (ECG's), stress tests, and cardiac angiograms, all of which are expensive. Risk evaluation screenings require blood samples, which are assessed alongside risk factors like tobacco use, diet, sleep disorders, physical inactivity, air pollution, and others.

More efficient, scalable, and non-invasive means of early detection could be used to trigger medical interventions, prompt preventive care by physicians, and/or engender behavioral change on the part of those prone to or suffering from CVD. Applying data mining techniques to CVD datasets to predict risk based on existing or easy-to-collect health data could improve healthcare outcomes and mortality rates.

The Cleveland dataset is the most complete CVD dataset and is the most frequently used in data science experimentation. It has a small number of cases (n = 303) but numerous variables including the 14 most commonly selected for analysis which were are the ones selected for this analysis as well (m = 14, including the target class). Small numbers of observations are common with health data given the costs of collecting experimental data and the privacy risk considerations of observational data.

1.2 Research Question and Approach

A multitude of approaches and methodologies have been attempted by researchers with the aim of predicting the presence of heart disease (the target class) based on the 13 other variables most commonly selected from the Cleveland dataset.

For this project, different classification techniques - including logistic regression, random forests, Naive Bayes, and Support Vector Machines models - that preceding researchers have found fruitful were evaluated, synthesized data was harnessed, and attempts were made to improve upon the models' performance.

2 DATA PREPARATION

2.1 Original Data Description

The original data set has 13 predictor variables, 8 of which are categorical and range from 2 to 5 levels while the other 5 are numeric. The target variable is a binary categorical variable that indicates whether or not the patient has heart disease with 1 indicating presence of heart disease and 0 indicating no heart disease. There were no missing values in the data set. Descriptions of each of the variables are provided in Table 1: Data Dictionary.

Table 1: Data Dictionary

VARIABLE	DEFINITION	TYPE
age	Age	continuous numerical predictor
sex	Sex. Female = 0, Male = 1	categorical predictor
cp	Chest pain type. Scale of 0 to 4	categorical predictor
trestbps	Diastolic blood pressure in mmHg	continuous numerical predictor
chol	Serum cholesterol (mg/dl)	continuous numerical predictor
fbs	Fasting blood sugar. Greater than 120mg/dl, value of 0 or 1	categorical predictor
restecg	Resting ECG. Value of 0, 1, or 2	categorical predictor
thalach	Maximum heartrate achieved from thallium test	continuous numerical predictor
exang	Exercise-induced angina. Value of 0 or 1	categorical predictor
oldpeak	Old-peak.ST depression induced by exercise relative to rest	continuous numerical predictor
slope	Slope of peak exercise ST segment, value of 1, 2, or 3	categorical predictor
ca	Number of major vessels (0-3) colored by fluoroscopy	categorical predictor
thal	Exercise thallium scintigraphic defects	categorical predictor
target	Response Variable - No Heart Disease = 0, Heart Disease = 1	categorical predictor

2.1.1 Original Data Summary Statistics

Table 2: Summary statistics for numeric variables in the original data set

	n	min	mean	median	max	sd
age	303	29	54.366337	55.0	77.0	9.082101
trestbps	303	94	131.623762	130.0	200.0	17.538143
chol	303	126	246.264026	240.0	564.0	51.830751
thalach	303	71	149.646865	153.0	202.0	22.905161
oldpeak	303	0	1.039604	0.8	6.2	1.161075

Table 3: Summary statistics for categorical variables in the original dataset

	sex	cp	ca	exang	fbs	restecg	slope	target	thal
	0: 96	0:143	0:175	0:204	0:258	0:147	0: 21	0:138	0: 2
	1:207	1: 50	1: 65	1: 99	1: 45	1:152	1:140	1:165	1: 18
	NA	2: 87	2: 38	NA	NA	2: 4	2:142	NA	2:166
	NA	3: 23	3: 20	NA	NA	NA	NA	NA	3:117
	NA	NA	4: 5	NA	NA	NA	NA	NA	NA

2.1.2 Original Data Summary Statistics Graphs

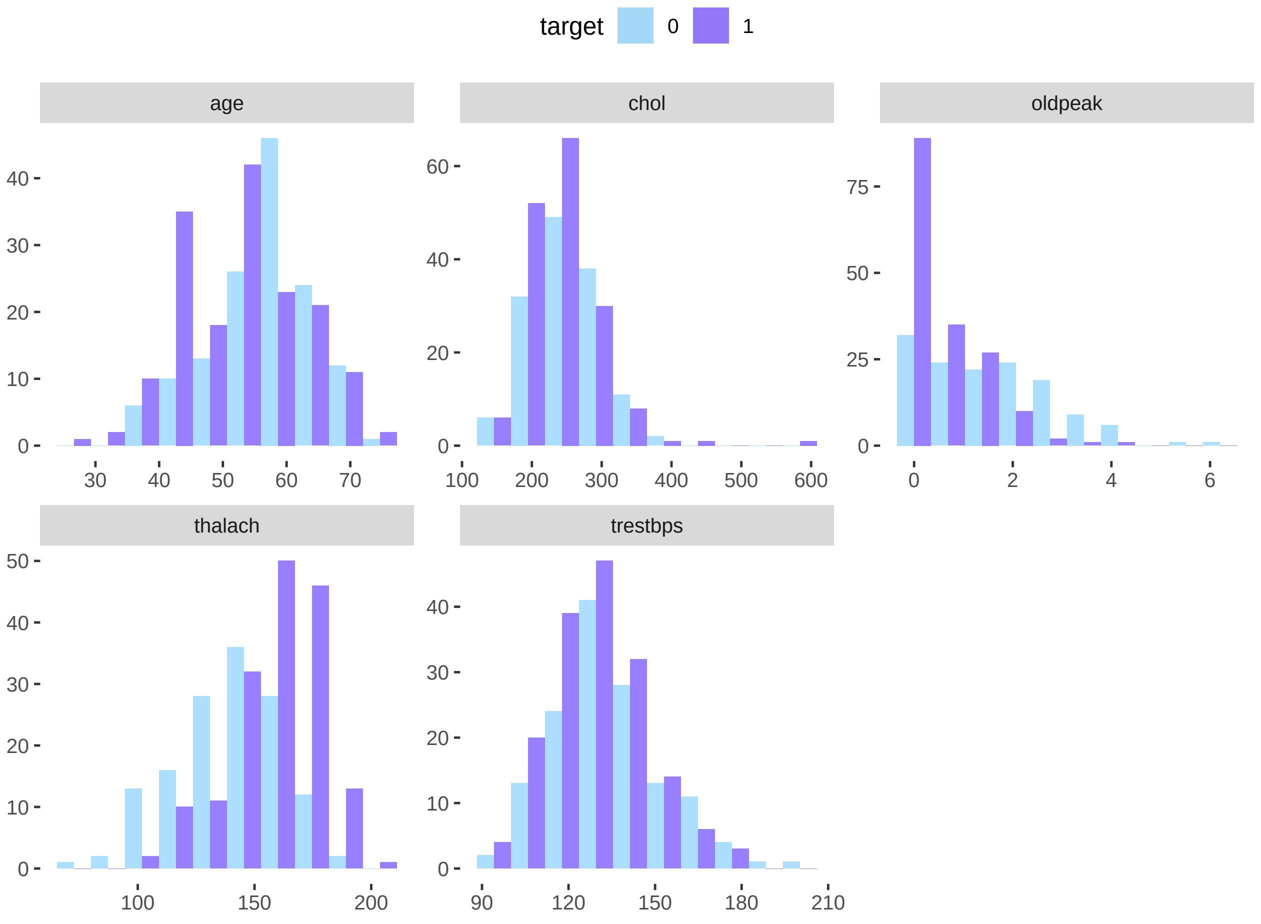


Figure 1: Numeric Data Distributions as a Function of TARGET

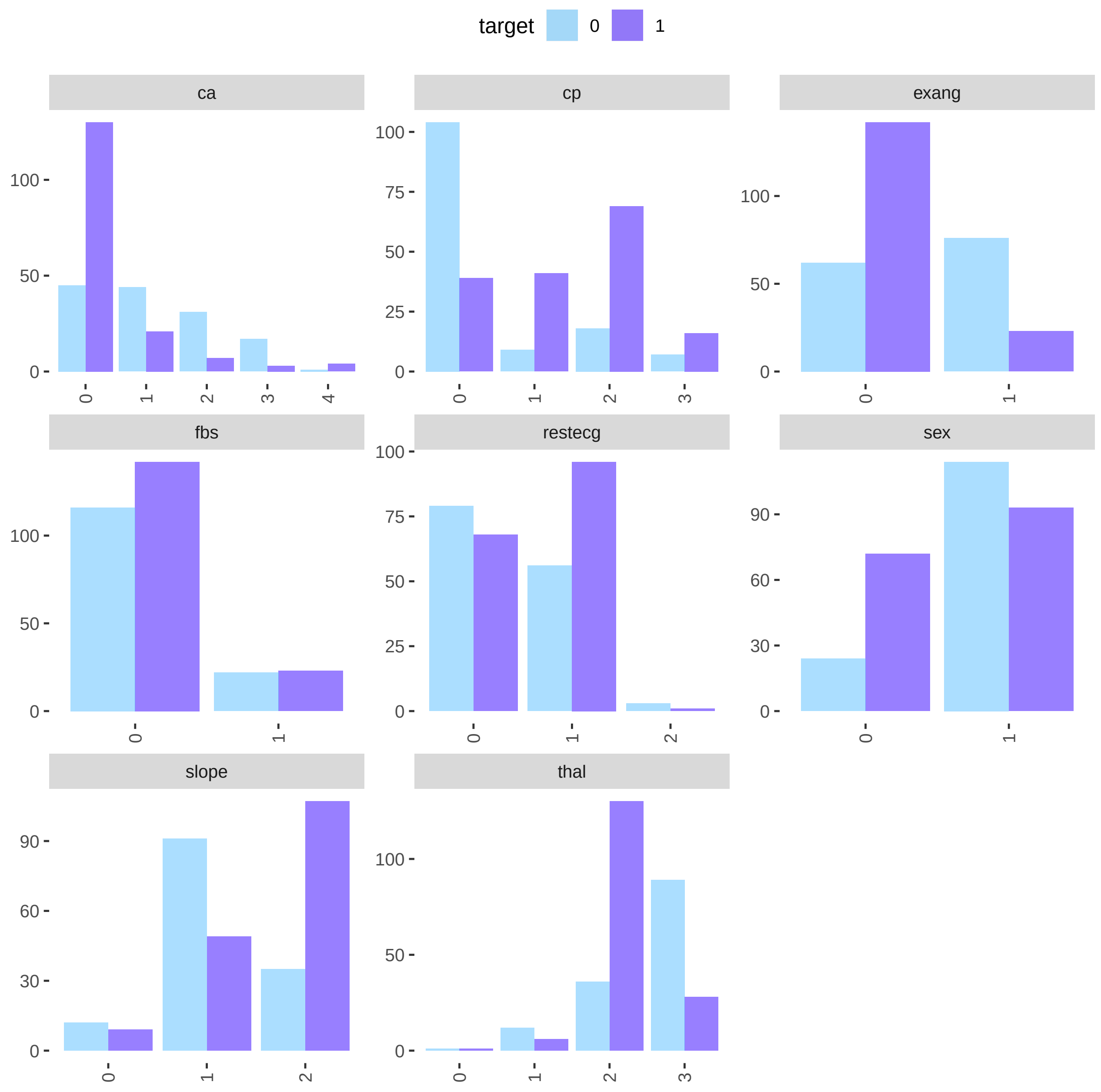


Figure 2: Categorical Data Distributions as a Function of TARGET

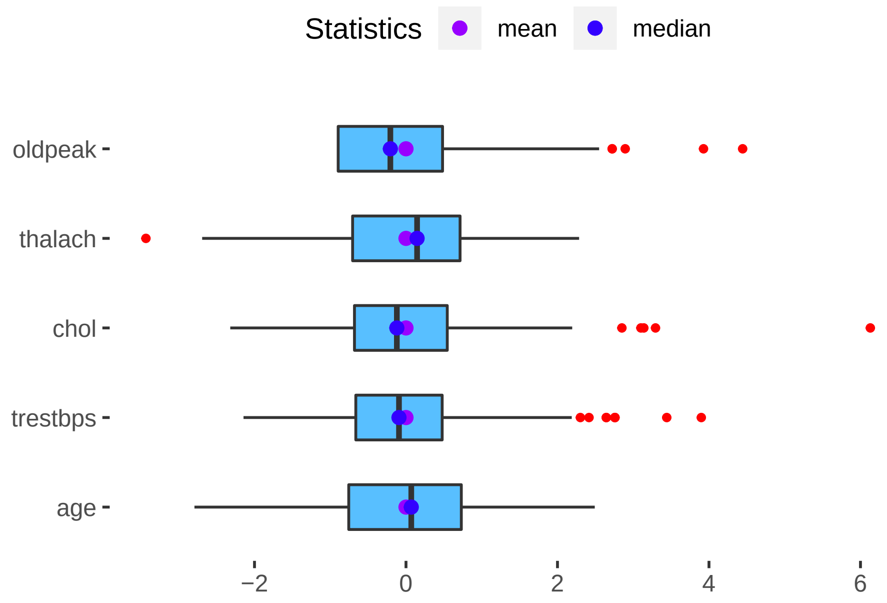


Figure 3: Scaled Boxplots for Numeric Variables

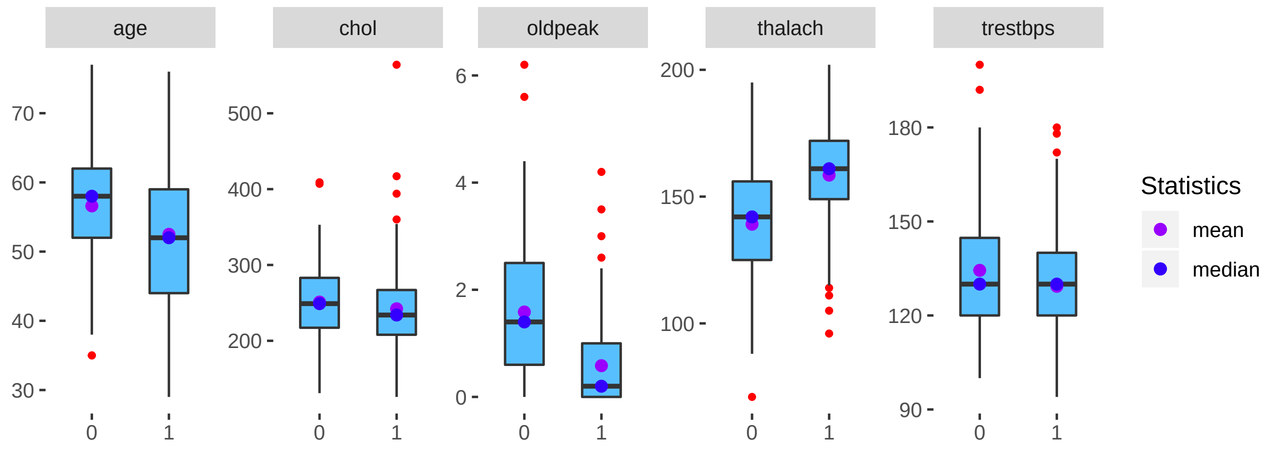
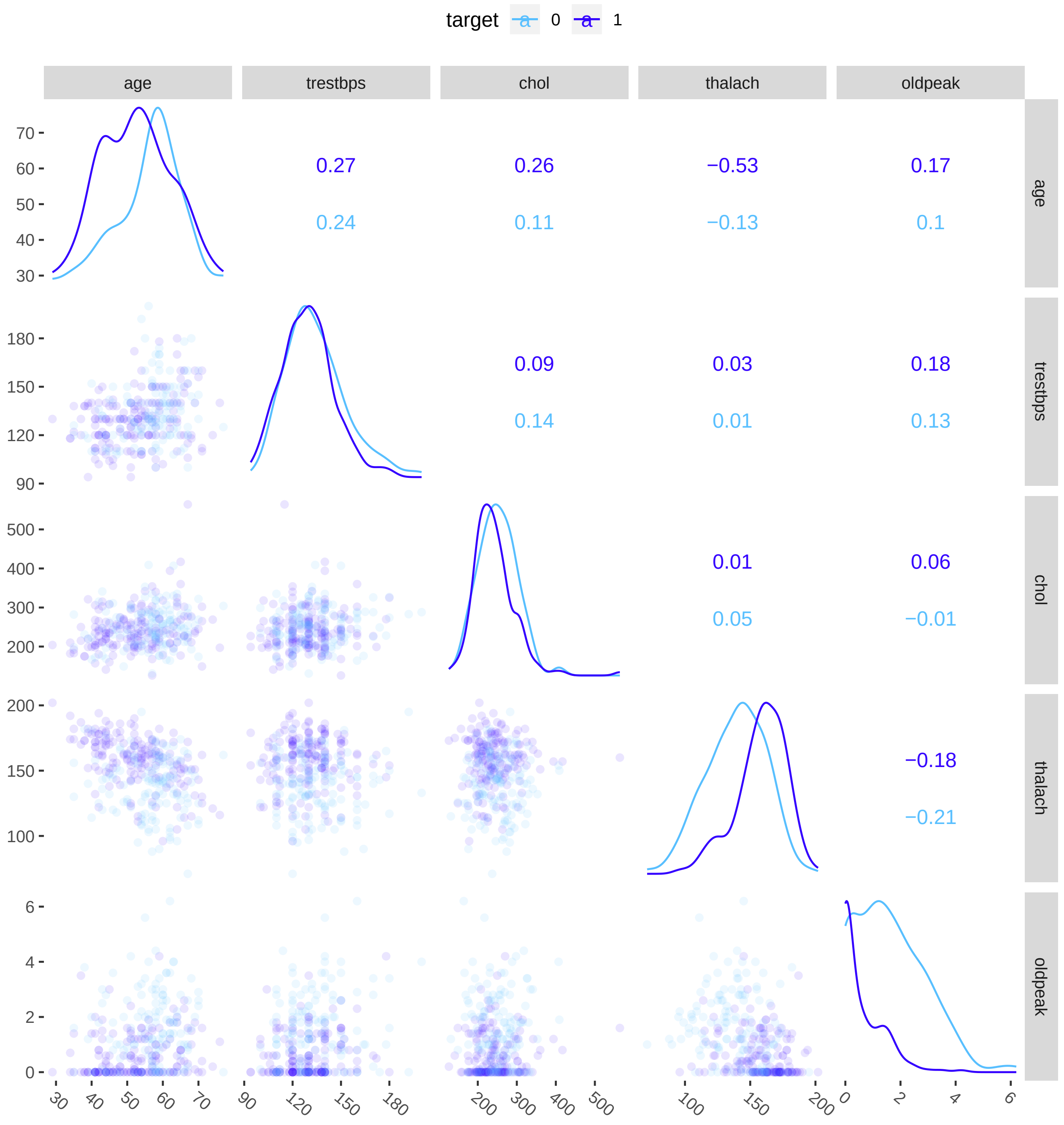


Figure 4: Linear relationship between each numeric predictor and the target



2.2 Cross-validation

Cross-validation is a resampling procedure to evaluate machine learning models. The in-sample data is divided randomly into equally-sized k groups, also known as folds. This approach involves randomly dividing the in-sample data into k groups or folds, of equal size. In this approach, the validation set is the first fold, and the remaining k-1 folds have the method fit across them. To use nested cross-validation, the outer cross-validation provides performance assessment obtainable using a method of constructing a model, including feature selection. In order to independently select the features in each fold of the outer cross-validation, the inner cross-validation is utilized. The evaluation score is retained and the model discarded to summarize the skill of the model using the evaluation scores.

One of the most popular cross validation techniques is Grid Search. Grid search experiments are common in the literature of empirical machine learning, where they are used to optimize the hyper-parameters of learning algorithms. It is common to perform multi-stage automated grid experiment, however, fine resolution for optimization would be computationally expensive. Grid search experiments allocates many trials to the exploration of dimensions that may not matter and suffer from poor coverage in dimensions that are important.

On the other hand, Random search found better models in most cases and required less computational time. It is easier to carry out and more practical in terms of statistical independence of every trial. The experiment can be stopped at any time and can be added without adjustment of grid or committing larger experiment as every trial can be carried out asynchronously.

To minimize the computation time, Coarse to Fine strategy was implemented. During the Coarse search phase, we use the random search technique, then, simply filter the under-performing parameter values out to run Finer search to find the best values for parameters.

2.3 Bootstraping ‘synthetic’ data

The idea of synthesizing data is to replace some or all observed values so that the statistical features of the original data are preserved. This approach can be used to anonymize data subjects, keep actual observations confidential, or comply with legal or regulatory requirements regarding identifiable information while still performing data modeling or other data-related tasks.

For our purposes, the motivation to synthesize data is to augment the size of the dataset, simulating a larger number of cases based on the original distributions of the observed data using the synthpop package in R. In this way, we can scale the data from hundreds to thousands or tens of thousands of cases with values for each variable. This enables a wider range of potential modeling techniques, and we are curious if it will allow models that can benefit from larger samples to achieve better or more stable performance.

2.3.1 Synthesis diagnostics

The original Cleveland dataset contains n = 303 observations. The synthesized dataset is simulated based on the same probability distribution, but is 20 times larger at n= 6,060.

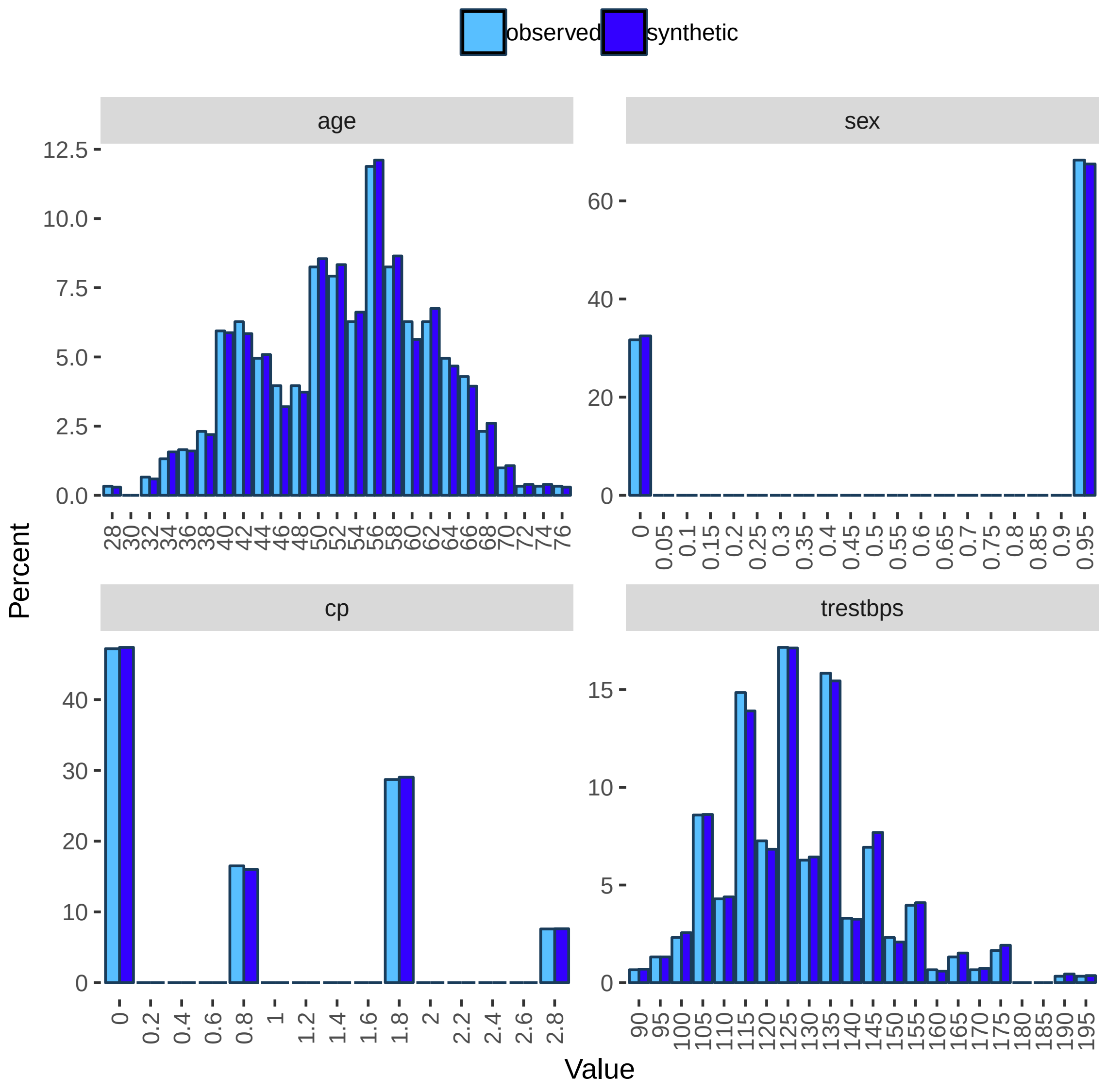


Figure 5: Data Distribution - Original vs. Synthesized

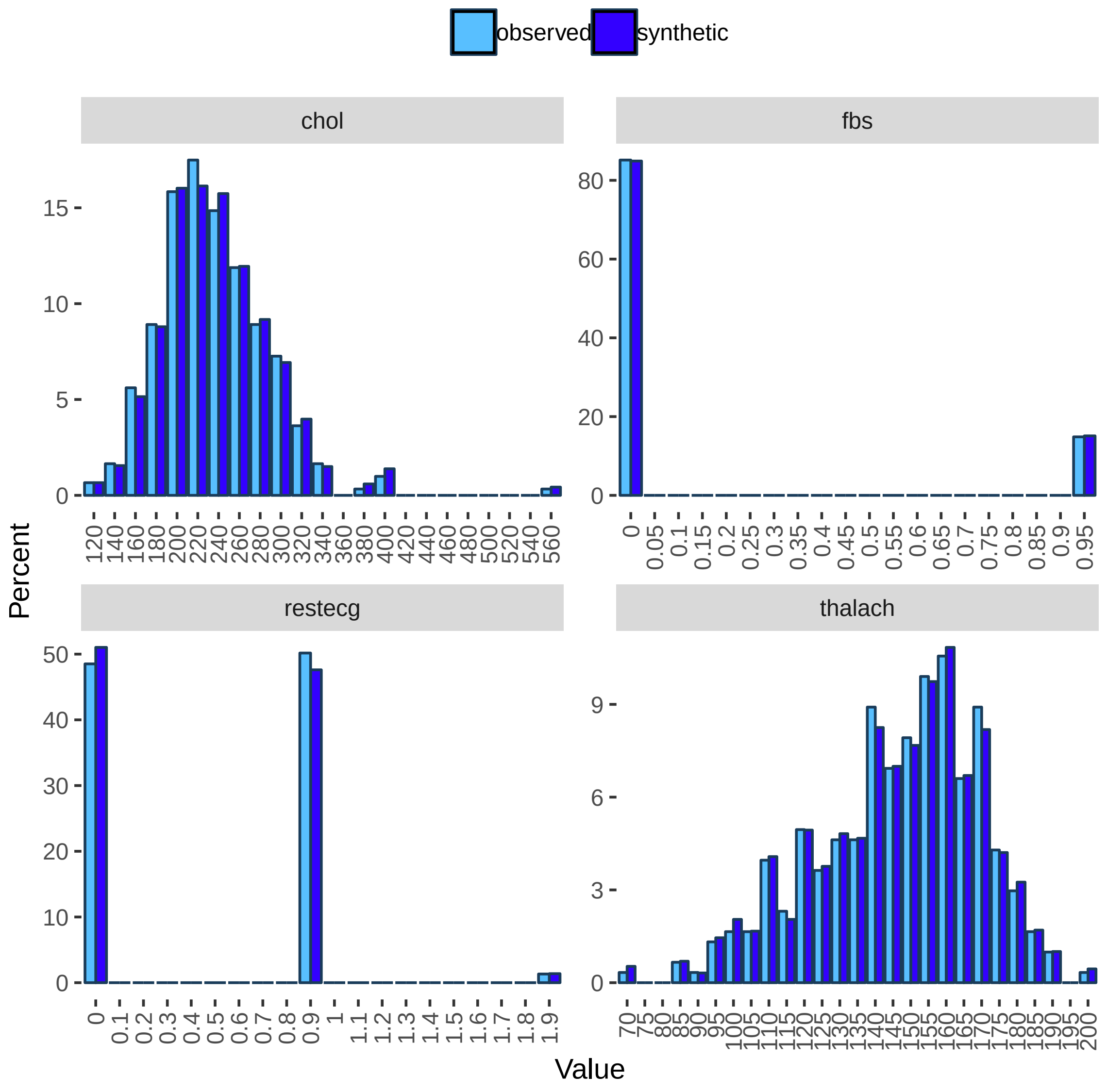


Figure 6: Data Distribution - Original vs. Synthesized

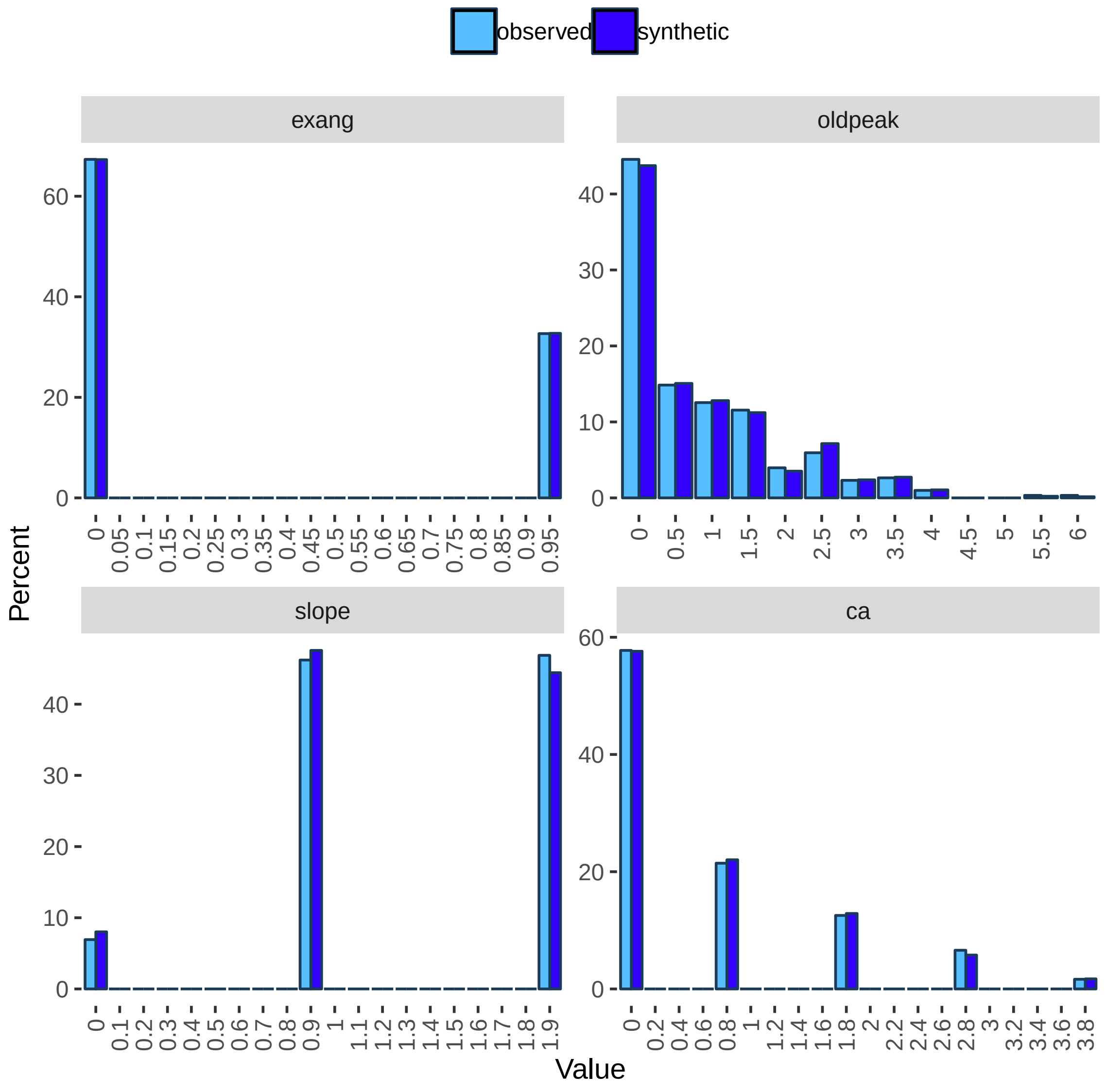


Figure 7: Data Distribution - Original vs. Synthesized

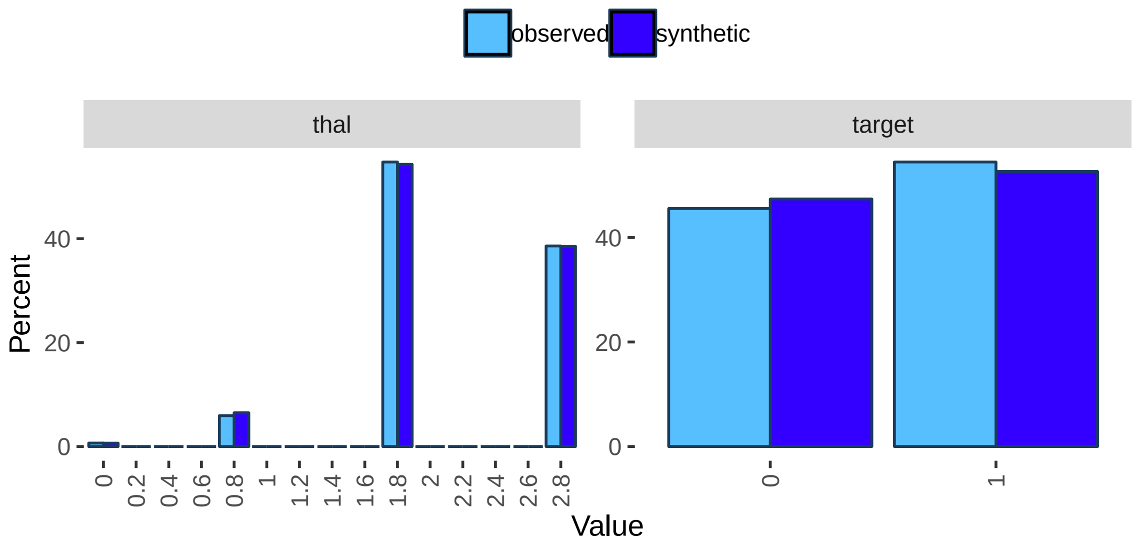


Figure 8: Data Distribution - Original vs. Synthesized

2.3.2 Synthesized data summary statistics

Table 4: Summary statistics for numerical variables in the synthesized dataset

	n	min	mean	median	max	sd
age	6060	29	54.48812	56.0	77.0	9.051595
trestbps	6060	94	132.04868	130.0	200.0	18.034574
chol	6060	126	248.28845	243.0	564.0	53.921330
thalach	6060	71	149.27921	152.0	202.0	23.459684
oldpeak	6060	0	1.05467	0.8	6.2	1.142782

Table 5: Summary statistics for categorical variables in the synthesized dataset

	sex	cp	ca	exang	fbs	restecg	slope	target	thal
	0:1968	0:2871	0:3491	0:4077	0:5145	0:3092	0: 486	0:2871	0: 39
	1:4092	1: 968	1:1335	1:1983	1: 915	1:2885	1:2882	1:3189	1: 393
	NA	2:1759	2: 780	NA	NA	2: 83	2:2692	NA	2:3292
	NA	3: 462	3: 350	NA	NA	NA	NA	NA	3:2336
	NA	NA	4: 104	NA	NA	NA	NA	NA	NA

3 BUILDING MODELS

Our literature review highlighted a wide array of approaches to classification taken to predict the presence of heart disease using the 13 variables most commonly selected from the Cleveland dataset.

Shouman et al. compiled an exhaustive review of over 60 papers published between 2000 and 2016 that detail different classification modeling approaches built on heart disease datasets, include the Cleveland dataset:

Summarizing model performance (median accuracy) by type:

We focused on a few pieces of research in particular:

- The aim of Shouman et al.'s work is to evaluate a potential low-cost heart disease expert system risk evaluation tool leveraging non-invasive data attributes. This is explored by evaluating the Cleveland dataset alongside another dataset from Canberra not available via the UCI machine learning repository. When constrained to Cleveland's non-invasive data attributes, the best performance is seen with a combination of age, sex, and resting blood pressure. This line of research also explores integrating K-means clustering with decision tree models to improve accuracy.
- Assari et al.'s broader data-mining focus finds that SVM and Naive Bayes outperform KNN (of K=7) and Decision Tree in terms of accuracy when using 10-fold cross-validation. Its results identify the most important features as chest pain type, exercise thallium, and coronary artery disease.
- Sabay et al. seek to assess the application of ML techniques requiring more observations to the Cleveland dataset so improve its generalizability. To that end, a surrogate synthetic dataset is bootstrapped using the Synthpop package in R. Logistic Regression is found to be more accurate and stable than Random Forest and Decision Tree methods, both for the original dataset as well as a 50,000-observation surrogate. An ANN perceptron model built on a 60,000-observation surrogate dataset achieves accuracy and recall above 95%.

We selected four modeling approaches based on our review of the literature and findings:

- Logistic Regression
- Random Forest
- Support Vector Machines
- Naive Bayes

3.1 Logistic regression

The logistic regression model involves picking one or more variables in comparison to the target. All of the available variables were initially compared to the target variable in both the original data and the synthetic data. The result of this comparison in the original data was that the patient's sex, chest pain type, and the number of vessels colored by fluoroscopy were the most influential of the variables. In the synthetic data, the major influencers were the patient's age, sex, chest pain type, serum cholesterol, and the number of vessels colored by fluoroscopy.

Multiple models were tested for their validity for each type of data. For the original data, the first model tested was looking at the individual influence of patient's sex, chest pain type, and the number of vessels colored by fluoroscopy on the target variable. The second model tested examined the relationship between the patient's sex and chest pain type, and how both of these variables interacted with each other, and then the number of vessels colored by fluoroscopy when compared with the target variable. The third model looked at all numeric variables compared to the target, and the last model looked at all categorical (factorized) variables compared to the target.

	R2	Adj..R2	AIC	BIC
Select 1	0.43	0.41	235.6967	270.6273
Select 2	0.44	0.41	237.9943	283.4041
Numeric	0.29	0.27	283.0819	307.5334
Factors	0.57	0.54	182.3636	248.7317

Across the original data, the model with the best preliminary performance was the factorized variables.

A similar process occurred with the synthetic data that resulted in three models instead of four. The first model compared the patient's age, sex, chest pain type, serum cholesterol, and the number of vessels colored by fluoroscopy to the target variable. The second model looked at all numeric variables compared to the target, and the last model looked at all categorical (factorized) variables compared to the target.

	R2	Adj..R2	AIC	BIC
Select	0.40	0.39	4605.077	4682.913
Numeric	0.18	0.18	6090.317	6135.722
Factors	0.42	0.42	4417.274	4540.515

The synthetic model proves once again the factorized variables are more influential than the others tested.

To further single out the best model, two kinds of predictions were made for each model: predictions against the test data for the data the model was created from, and predictions against the test data for the data the model was not created from.

	Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue	Mcnemar
factors_sXo	0.8933333	0.7847920	0.8005983	0.9528100	0.5466667	0.0000000	1.0
factors_o	0.8266667	0.6546227	0.7218513	0.9043494	0.5466667	0.0000003	0.2
select_sXo	0.8266667	0.6511628	0.7218513	0.9043494	0.5466667	0.0000003	1.0
factors_s	0.8110964	0.6186257	0.7904530	0.8305254	0.5264201	0.0000000	0.0
select1_o	0.8000000	0.6014878	0.6916739	0.8835179	0.5466667	0.0000041	0.3
select2_o	0.8000000	0.6014878	0.6916739	0.8835179	0.5466667	0.0000041	0.3
factors_oXs	0.7912814	0.5808659	0.7699249	0.8115017	0.5264201	0.0000000	0.3
select_s	0.7701453	0.5384219	0.7481091	0.7911285	0.5264201	0.0000000	0.3
select1_oXs	0.7516513	0.5026777	0.7290824	0.7732397	0.5264201	0.0000000	0.2
select2_oXs	0.7490092	0.4975275	0.7263687	0.7706797	0.5264201	0.0000000	0.1
numeric_o	0.7200000	0.4278968	0.6043696	0.8175946	0.5466667	0.0015702	0.3
numeric_sXo	0.7066667	0.4051911	0.5902167	0.8061907	0.5466667	0.0033625	0.8
numeric_oXs	0.6737120	0.3400535	0.6494451	0.6973032	0.5264201	0.0000000	0.0
numeric_s	0.6644650	0.3234520	0.6400490	0.6882413	0.5264201	0.0000000	0.0

The most accurate model for the original data and the synthetic data was the factorized model. The most accurate model for data not matching what the model was built on was also the factorized model. The best of the best was definitely the factorized model for the original data.

	0	**1**
0	30	9
1	4	32

Figure 9: Confusion Matrix for Factorized Model on Original Data

3.2 Random forest

Random Forest consists of numerous decision trees that are generated based on bootstrap sampling from the in-sample data. Subsampling reduces the variance of trees substantially and the random feature selection decorrelates them to improve the predictive accuracy and control over-fitting.

The bootstrap resampling of the data for training each tree increases the diversity between the trees. Each tree is composed of a root node, branch nodes, and leaf nodes. For each node of a tree, the optimal node splitting feature is selected from a set of features that are again randomly selected. The final output is an ensemble of random forest trees, so that classification can be performed via majority vote.

Tuning hyperparameter values in the model significantly impact the accuracy of model performance. The three parameters that we tune in this experiment are the number of trees, maximum depth, and the number of features to randomly select. We use a coarse to fine search strategy for hyperparameter tuning. Our model used normally distributed random values for parameters for a few tries. Then, we incorporate cross-validation during the training process to evaluate the results. Upon completion, the parameters from the first pass models are used to create a refined range for parameter selection.

3.2.1 Hyper parameter tuning

3.2.1.1 Tune using caret

The caret package in R provides an excellent facility to tune machine learning algorithm parameters. Not all machine learning algorithms are available in caret for tuning. The choice of parameters is left to the developers of the package. Only those algorithm parameters that have a large effect are available for tuning in caret. As such, only `mtry` parameter is available in caret for tuning. The reason is its effect on the final accuracy and that it must be found empirically for a dataset. The `ntree` parameter is different in that it can be as large as you like, and continues to increases the accuracy up to some point. It is less difficult or critical to tune and could be limited more by compute time available more than anything.

3.2.1.1.1 Random Search

One search strategy that we can use is to try random values within a range. This can be good if we are unsure of what the value might be and we want to overcome any biases we may have for setting the parameter (like the suggested equation above). Let us try a random search for `mtry` using caret: We can see that the most accurate value for mtry was 2 with an accuracy of 0.8084378

3.2.1.1.2 Grid Search

Grid search experiments are common in the literature of empirical machine learning, where they are used to optimize the hyper-parameters of learning algorithms. It is common to perform multi-stage automated grid experiment, however, fine resolution for optimization would be computationally expensive. Grid search experiments allocates many trials to the exploration of dimensions that may not matter and suffer from poor coverage in dimensions that are important. We can see that the most accurate value for `mtry` was 1 with accuracy of 0.8158102

3.2.1.2 Tune Using Algorithm Tools

Some algorithms provide tools for tuning the parameters of the algorithm. For example, the random forest algorithm implementation in the randomForest package provides the tuneRF() function that searches for optimal mtry values given your data. We can see that the most accurate value for mtry was 2 with an OOBError of 0.1650165 This does not really match up with what we saw in the caret repeated cross validation experiment above, Nevertheless, it is an alternate way to tune the algorithm.

3.2.1.3 Craft your own parameter search

3.2.1.3.1 Tune Manually

We want to keep using caret because it provides a direct point of comparison to our previous models and because of the repeated cross validation test harness that we like as it reduces the severity of overfitting. One approach is to create many caret models for our algorithm and pass in a different parameters directly to the algorithm manually. Let's look at an example doing this to evaluate different values for `ntree` while holding mtry constant.

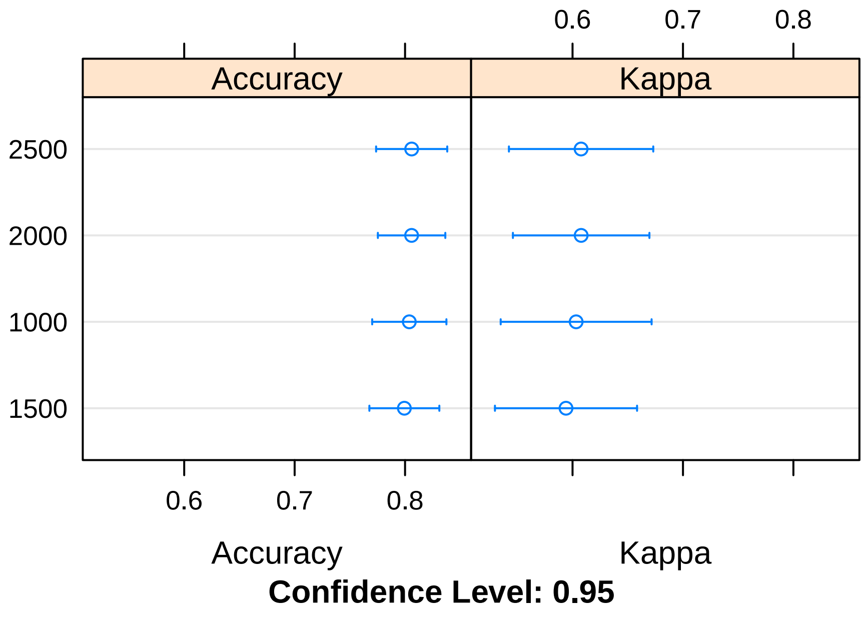


Figure 10: Random Forest Hyperparameter Tuning Manual Search

We can see that the most accuracy value for ntree was perhaps 1000 with a mean accuracy of 80.38% (a lift over our very first experiment using the default mtry value). The results perhaps suggest an optimal value for ntree between 2000 and 2500. Also note, we held mtry constant at the default value. We could repeat the experiment with a possible better mtry=2 from the experiment above, or try combinations of of ntree and mtry in case they have interaction effects.

3.2.1.3.2 Extend Caret

Another approach is to create a "new" algorithm for caret to support. This is the same random forest algorithm you are using, only modified so that it supports multiple tuning of multiple parameters. A risk with this approach is that the caret native support for the algorithm has additional or fancy code wrapping it that subtly but importantly changes it's behavior. You many need to repeat prior experiments with your custom algorithm support. We can define our own algorithm to use in caret by defining a list that contains a number of custom named elements that the caret package looks for, such as how to fit and how to predict. See below for a definition of a custom random forest algorithm for use with caret that takes both an mtry and ntree parameters. Now, let's make use of this custom list in our call to the caret train function, and try tuning different values for ntree and mtry.

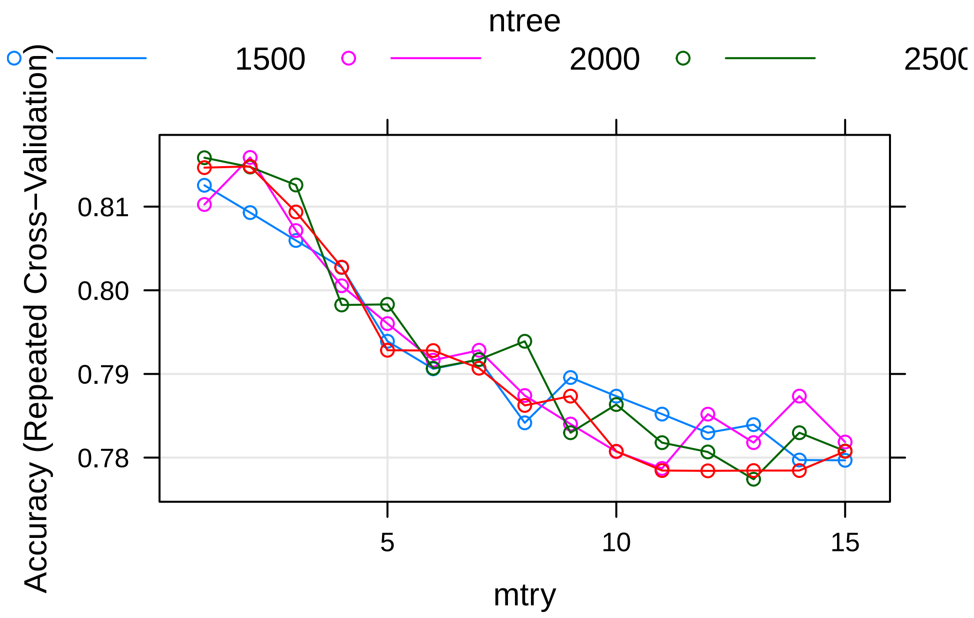


Figure 11: Random Forest Hyperparameter Tuning Custom Search

You can see that the most accurate values for ntree and mtry were 1500 and 2 with an accuracy of 84.43%. We do perhaps see some interaction effects between the number of trees and the value of ntree.

3.2.2 Random Forest Baseline Model

we will stick to tuning two parameters, the `mtry` and the `ntree` that have the most influence on accuracy of RF model. `mtry` is the number of variables randomly sampled as candidates at each split. `ntree` is the number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times. Let us create a baseline for comparison, using recommended default values for parameters: `mtry` floor(sqrt(ncol(x))) and `ntree` 500.

The baseline model used total 303 samples which includes 13 predictor variables to classify two classes in target. Resampling was done using cross-validation for 10 fold, repeating 3 times. The estimated accuracy is 0.7962489

3.2.3 Random Forest Final Model And Evaluation

The final model is developed by using generated synthetic data with the help of minority class data. Simply put, it takes the minority class data points and creates new data points which lie between any two nearest data points joined by a straight line. In order to do this, the algorithm calculates the distance between two data points in the feature space, multiplies the distance by a random number between 0 and 1 and places the new data point at this new distance from one of the data points used for distance calculation. Note the number of nearest neighbors considered for data generation is also a hyperparameter and can be changed based on requirement.

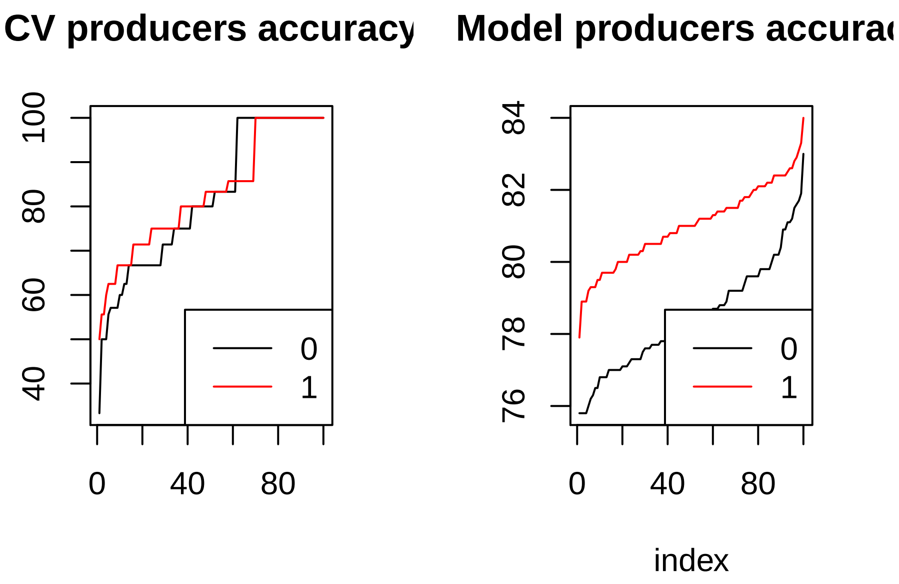


Figure 12: RF Classifier cross-validation using original data

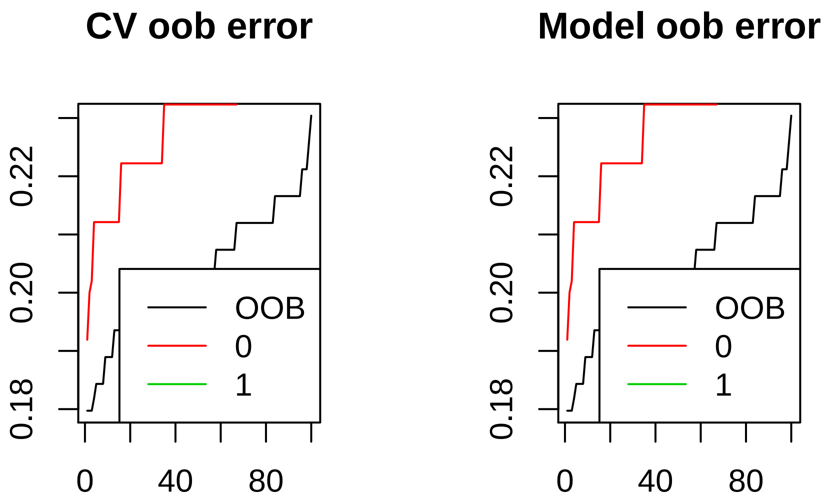


Figure 13: RF Classifier cross-validation OOB error using original data

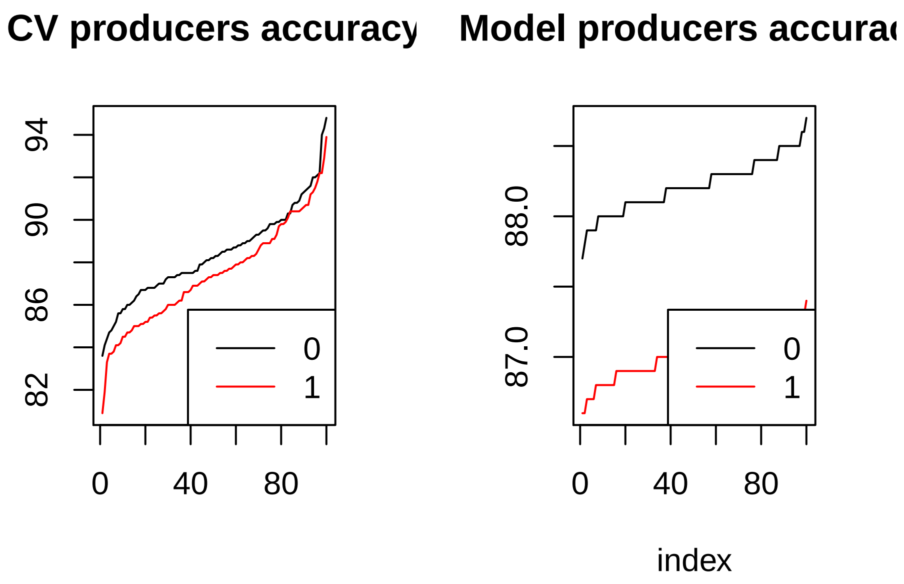


Figure 14: RF Classifier cross-validation using synthesized data

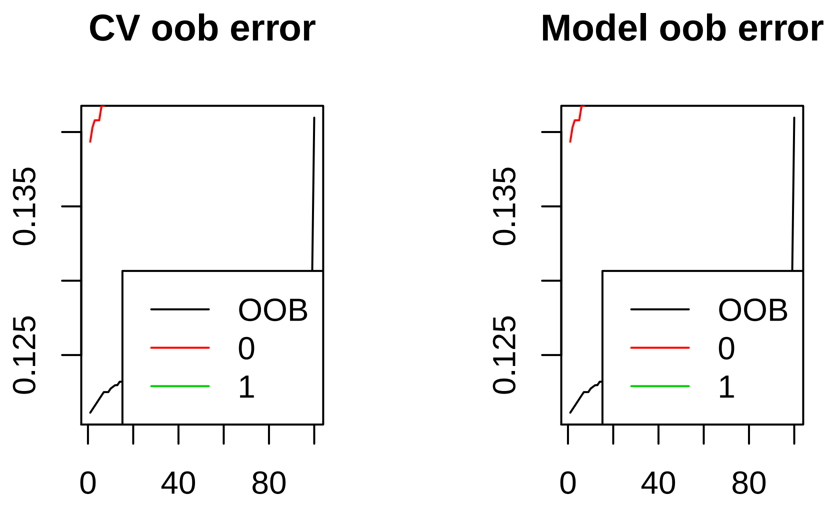


Figure 15: RF Classifier cross-validation OOB using synthesized data

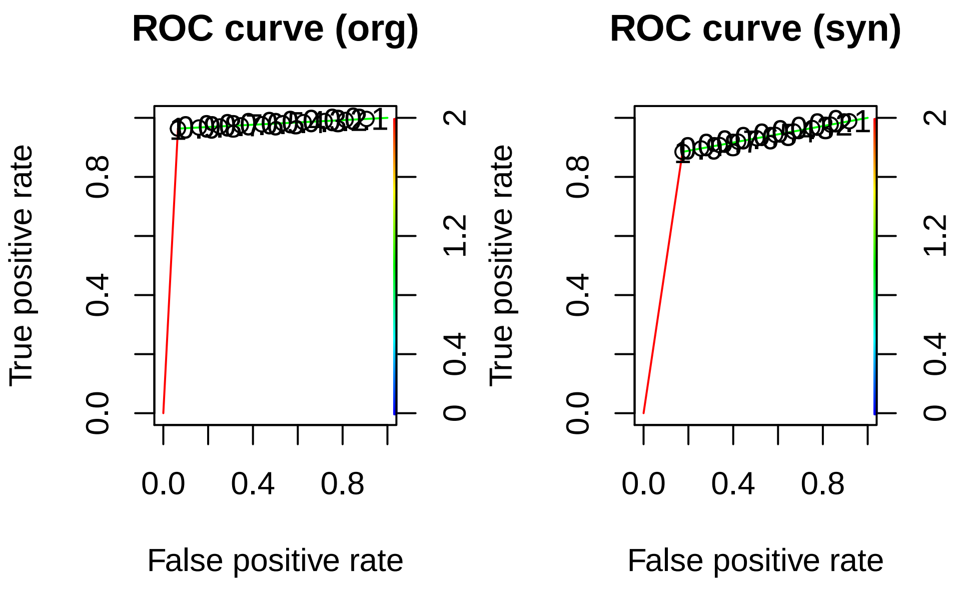


Figure 16: RF classifier ROC curve Original(left), Synthesized(right)

The chart shows accuracy and oob error graph of RF classifiers built on original data, synthesized data on top and bottom row respectively. It appears that the classifier built on original model shows higher OOB error despite the higher accuracy rate. This is due to classifier having `seen` the test data when building the model.

3.3 Support Vector Machines

Per our literature review, Assari et al found that Support Vector Machine (SVM) models provided better accuracy than Naive Bayes and Decision Tree approaches.

SVM models are a form of supervised learning used to create discriminative large-margin classifiers, defining a decision boundary between classes based on a hyperplane. This hyperplane maximizes the margin, or distance, between the nearest points of different labeled classes while prioritizing correct classification. The classifier can then be used to predictively categorize unlabeled examples.

SVM models perform well in high dimensions and can be used to produce linear as well as non-linear classification depending on the kernel function employed. However, they are prone to overfitting and can be hard to interpret, particularly when in higher dimensions.

3.3.1 Kernel Methods

Kernels are algorithms that operate in high-dimensional feature space without explicitly mapping data to the coordinates of that space. Mathematically, this 'kernel trick' means scalar solutions to dot products can be used without computationally laborious transformations.

Practically for SVM models, if we can map the feature space to higher dimensions in which the classes of cases become separable, then a linear classifier can be used to solve non-linear problems.

As kernel values depend on the inner products of feature vectors, it's best practice to scale variables to range of [0, 1] or [-1, 1], which can be accomplished with the preProcess argument of caret's train function.

Choice of kernel function has a significant impact on the outcome of an SVM model. While this provides flexibility it also create potential points of failure, and SVM models are very sensitive to selection of kernel parameters.

Hsu et al recommend commencing with the Guassian radial basis function (RBF) kernel, which can handle nonlinear relationships, has fewer hyperparameters, and prevents fewer numerical challenges. For this project, we evaluated both RBFs and linear kernels, setting them via the method argument of caret's train function.

3.3.2 Tuning

A hard margin constraint prevents the classifier from allowing the margin to overlap with values. This constraint can be relaxed, which reduces variance but adds bias. By manipulating the regularization, or penalty, parameter C we can decrease (soft margin) or increase (hard margin) the weight of values within the margin on overall error. To test the impact of different C values we employ a grid search using the expand.grid function.

The tuning parameter sigma also impacts model fit. A smaller sigma tends to yield a local classifier with less bias and more variance, while a larger sigma value tends to yield a general classifier with more bias and less variance.

3.3.3 Approach

SVM models can be computationally intensive, so we first build the model and evaluate performance based on the original dataset of 303 cases before attempting with a larger synthesized dataset. We harness different kernel functions and tuned parameters to optimize performance based on accuracy (number of correct predictions as percentage of total cases) and kappa statistics (comparing observed and expected accuracy based on random chance, or interrater reliability).

3.3.4 Models

For the first SVM model, the original dataset is preprocessed (centered and scaled) before applying an RBF kernel with cross-validation (10 folds, 5 repeats). caret identifies the optimal model based on ROC, which is highest for sigma = .033 and C = .25. This model yields an accuracy of .813 and a kappa statistic of .622.

We generate a second SVM model with same preprocessing, RBF kernel, and cross-validation as the first, tuning it via a grid search around the sigma and C values. This second model is optimal on ROC at a lower sigma value (.01) and higher regularization parameter (.4), displaying more variance and a harder decision boundary, but lower accuracy (.787) and kappa (.565).

As tuning did not improve accuracy, we apply the same approach and parameters of the first model (identical preprocessing, RBF kernel, and cross-validation) to the synthesized data to create a third SVM model. Based on ROC, caret selects the optimal model with highest sigma = .031 and C = 1. This model performs better than both preceding on accuracy (.84) and kappa (.672). Compared with the first model, sensitivity has declined (.765 vs. .735), meaning slightly more false negatives; but specificity has increased (.854 vs. .927), for fewer false positives.

For a contrasting approach, we also trial a linear classifier (in place of the radial used in the first three models) using the synthesized data to create a fourth SVM model. This model achieves better accuracy (.827) and kappa statistic (.65) than the first and second models but does not outperform the third.

In summary, an SVM model using a radial kernel atop the larger synthesized dataset achieved an accuracy of .84, with a stricter penalties on false positives than false negatives. Given the model is intended as a diagnostic health tool, the optimal balance between sensitivity and specificity could be further explored (i.e. chance of misdiagnosing a patient at higher risk vs. cost of needlessly testing a patient with lower risk).

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  X0  X1
##          X0 576 104
##          X1 141 693
##
##                Accuracy : 0.8382
##                  95% CI : (0.8186, 0.8564)
##     No Information Rate : 0.5264
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6746
##
##  Mcnemar's Test P-Value : 0.02145
##
##             Sensitivity : 0.8033
##             Specificity : 0.8695
##          Pos Pred Value : 0.8471
##          Neg Pred Value : 0.8309
##              Prevalence : 0.4736
##          Detection Rate : 0.3804
##    Detection Prevalence : 0.4491
##       Balanced Accuracy : 0.8364
##
##        'Positive' Class : X0
##
```

3.4 Naive Bayes

A Naive Bayes classifier assumes that the presence (or absence) of a particular feature is unrelated to the presence (or absence) of any other feature. It considers that all variables independently contribute to the probability of heart disease. In spite of such simple assumptions and design, naive Bayes classifiers often work well in real-world situations. Additionally, they require relatively small amount of training data to estimate parameters.

Our first model was built on the original dataset, and our second on the synthetic dataset. In both cases, the models included all categorical, factorized variables. The numeric variables `age` and `sex` were removed to improve model accuracy, and the `chol` variable was converted into a categorical. Accuracy (.786 vs. .787) did not improve when the model was run on the larger synthetic dataset. A rise in sensitivity (.677 vs .751) is accompanied by a decline in specifity (.878 vs. .815), so fewer false negatives at the cost of more false positives.

The Naive Bayes classifier is often hard to beat in terms of CPU and memory consumption, and in certain cases its performance approximates more complicated, slower techniques.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  X0  X1
##         X0 534 159
##         X1 183 638
##
##                Accuracy : 0.7741
##                  95% CI : (0.7522, 0.795)
##     No Information Rate : 0.5264
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5462
```

```
##
##  Mcnemar's Test P-Value : 0.2136
##
##             Sensitivity : 0.7448
##             Specificity : 0.8005
##          Pos Pred Value : 0.7706
##          Neg Pred Value : 0.7771
##              Prevalence : 0.4736
##          Detection Rate : 0.3527
##    Detection Prevalence : 0.4577
##       Balanced Accuracy : 0.7726
##
##        'Positive' Class : X0
##
```

4 MODEL REVIEW AND SELECTION

4.1 Comparison of performance between models

The logistic regression model for the original data had an accuracy of 0.813. The random forest model took the lead with an accuracy on the original data of 0.951. It maintained its lead when the support vector machines model presented with an accuracy of 0.813, just as the logistic regression model had. Lastly, the weakest of the models was the Naive Bayes model with an accuracy of 0.787 on the original data. This exact pattern also held true for the synthetic data, except with a single blip - the support vector machines model was more efficient with the synthetic data (0.840) than it was with the original data.

4.2 Comparison of performance viz. other studies

When compared to the pre-existing models, the models produced for this project were on the whole sub-optimal save one: the random forest model. The random forest model managed to produce an accuracy that beat the best accuracy of the pre-existing models, 0.814. The logistic regression and the Naive Bayes models failed to meet or beat the average of the pre-existing models, while the support vector machine model exceeded the median but did not meet the highest pre-existing model's accuracy.

5 CONCLUSION

The previous efforts of researchers provide valuable information and insight into creating the best models possible for identifying a target variable in some data set. This is especially true of the Cleveland Heart Disease dataset. The accuracy of the models created before this project were unmatched until the creation of the random forest model provided in this document. This model, with its nearly 95% accuracy, could potentially help identify heart disease in individuals quickly and accurately.

6 APPENDIX

6.1 Supplemental tables and figures

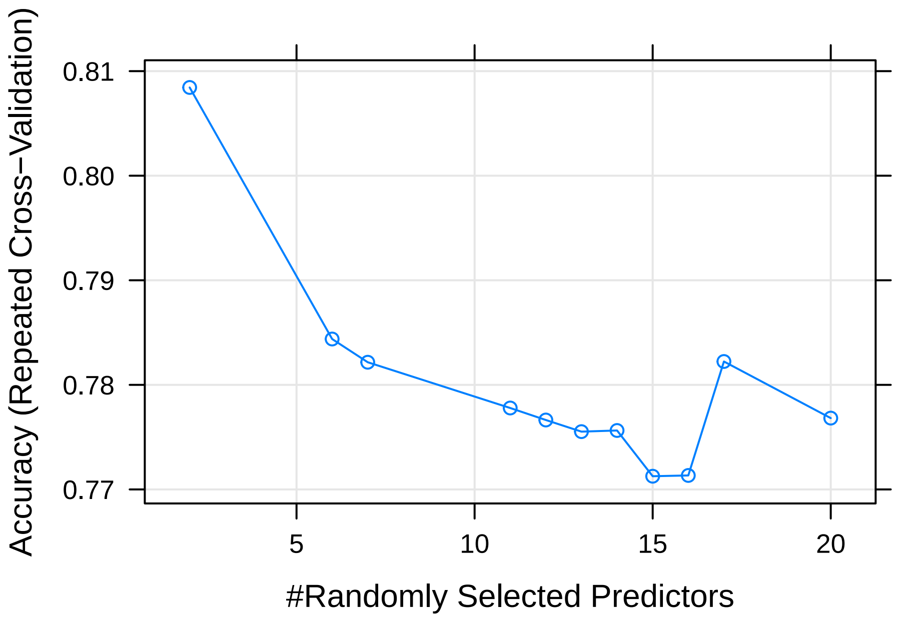


Figure 17: Random Forest Hyperparameter Tuning - Random Search

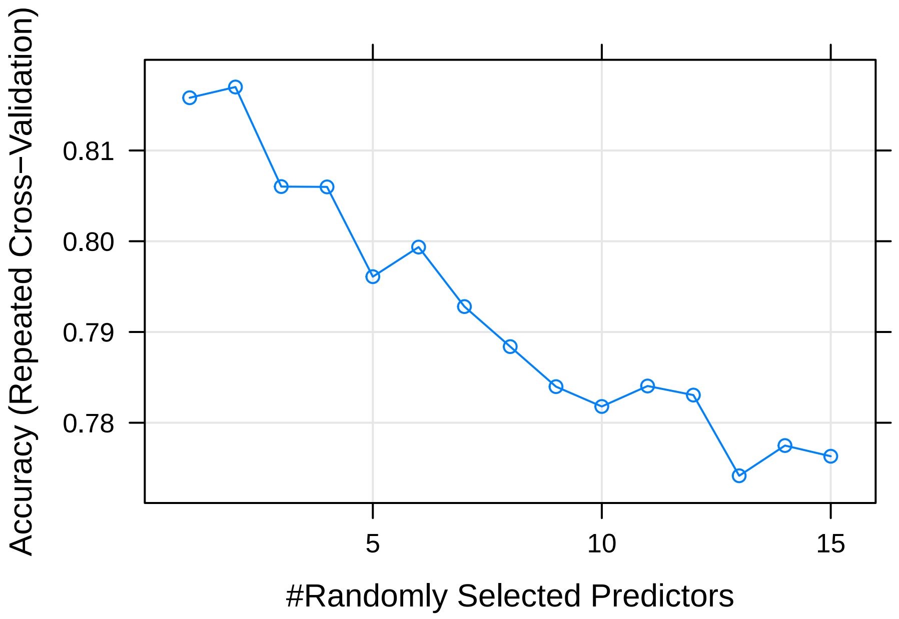


Figure 18: Random Forest Hyperparameter Tuning - Grid Search

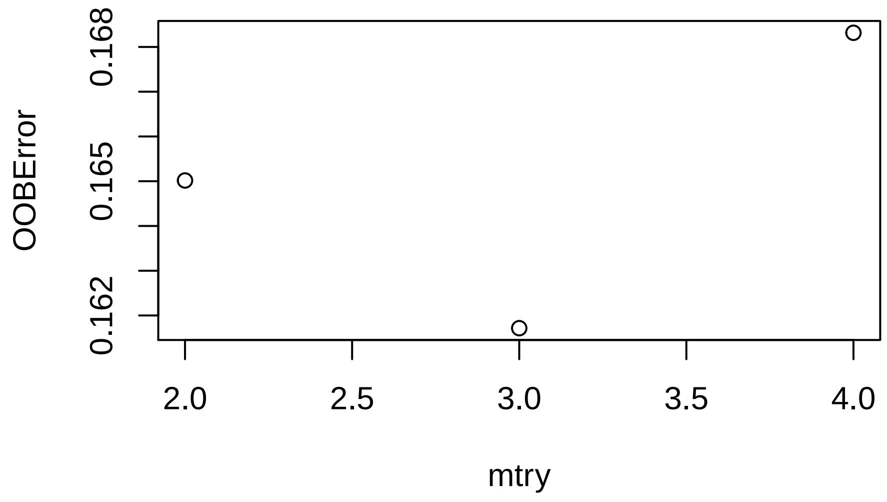


Figure 19: Random Forest Best mtry search

6.2 R statistical programming code

The appendix is available as script.R file in `projectFinal_heart` folder.

https://github.com/betsyrosalen/DATA_621_Business_Analyt_and_Data_Mining

```
### PLEASE ADD NEW PACKAGES IN ALPHABETICAL ORDER SO WE DON'T DUPLICATE LINES! ###
if (!require('car')) (install.packages('car'))
if (!require('caret')) (install.packages('caret'))
if (!require('caTools')) (install.packages('caTools'))
if (!require('corrplot')) (install.packages('corrplot'))
if (!require('data.table')) (install.packages('data.table'))
if (!require('DataExplorer')) (install.packages('DataExplorer'))
if (!require('doMC')) (install.packages('doMC'))
if (!require('dplyr')) (install.packages('dplyr'))
if (!require('e1071')) (install.packages('e1071'))
if (!require('faraway')) (install.packages('faraway'))
if (!require('GGally')) (install.packages('GGally'))
if (!require('ggfortify')) (install.packages('ggfortify'))
if (!require('ggplot2')) (install.packages('ggplot2'))
if (!require('gridExtra')) (install.packages('gridExtra'))
if (!require('huxtable')) (install.packages('huxtable'))
if (!require('jtools')) (install.packages('jtools'))
if (!require('kableExtra')) (install.packages('kableExtra'))
if (!require('kernlab')) (install.packages('kernlab'))
if (!require('MASS')) (install.packages('MASS'))
if (!require('mlbench')) (install.packages('mlbench'))
if (!require('pROC')) (install.packages('pROC'))
if (!require('pscl')) (install.packages('pscl'))
if (!require('psych')) (install.packages('psych'))
if (!require('randomForest')) (install.packages('randomForest'))
if (!require('reshape2')) (install.packages('reshape2'))
if (!require('rfUtilities')) (install.packages('rfUtilities'))
if (!require('ROCR')) (install.packages('ROCR'))
if (!require('rstan')) (install.packages('rstan'))
if (!require('synthpop')) (install.packages('synthpop'))
if (!require('tidyverse')) (install.packages('tidyverse'))
if (!require('tidyr')) (install.packages('tidyr'))

# REFERENCE STUDIES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

studies <- rbind(c('Hall',  '2000', 'Na?ve Bayes', '.832'),
                 c('Hall',  '2000', 'K Nearest Neighbour', '.821'),
                 c('Hall',  '2000', 'Decision Tree', '.753'),
                 c('Yan, Zheng et al.', '2003', 'Multilayer Perceptron', '.636'),
                 c('Herron', '2004', 'Support Vector Machine', '.836'),
                 c('Herron', '2004', 'J4.8 Decision Tree', '.776'),
                 c('Herron', '2004', 'Support Vector Machine', '.834'),
                 c('Andreeva', '2006', 'Na?ve Bayes', '.786'),
                 c('Andreeva', '2006', 'Decision Tree', '.757'),
                 c('Andreeva', '2006', 'Neural Network', '.828'),
                 c('Andreeva', '2006', 'Sequential Minimal Optimization', '.841'),
                 c('Andreeva', '2006', 'Kernel Density', '.844'),
                 c('Polat , Sahan et al.', '2007', 'Fuzzy-AIRS-K-Nearest Neighbour', '.870'),
                 c('Palaniappan and Awang', '2007', 'Na?ve Bayes', '.950'),
                 c('Palaniappan and Awang', '2007', 'Decision Tree', '.949'),
                 c('Palaniappan and Awang', '2007', 'Neural Network', '.935'),
                 c('De Beule, Maesa et al.', '2007', 'Artificial Neural Network', '.820'),
                 c('Tantimongcolwat, Naenna et al.', '2008', 'Direct Kernel Self-organizing Map', '.804
                 c('Tantimongcolwat, Naenna et al.', '2008', 'Multilayer Perceptron', '.745'),
                 c('Hara and Ichimura', '2008', 'Automatically Defined Groups', '.678'),
                 c('Hara and Ichimura', '2008', 'Immune Multi-agent Neural Network', '.823'),
                 c('Sitar-Taut, Zdrenghea et al.', '2009', 'Na?ve Bayes', '.620'),
                 c('Sitar-Taut, Zdrenghea et al.', '2009', 'Decision Trees', '.604'),
                 c('Tu, Shin et al.', '2009', 'Bagging Algorithm', '.814'),
                 c('Das, Turkoglu et al.', '2009', 'Neural Network Ensembles', '.890'),
                 c('Rajkumar and Reena', '2010', 'Na?ve Bayes', '.523'),
                 c('Rajkumar and Reena', '2010', 'K Nearest Neighbour', '.457'),
                 c('Rajkumar and Reena', '2010', 'Decision List', '.520'),
                 c('Srinivas, Rani et al.', '2010', 'Na?ve Bayes', '.841'),
                 c('Srinivas, Rani et al.', '2010', 'One Dependency Augmented Na?ve Bayes', '.805'),
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Back-propagation Neural Network', '
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Bayesian Neural Network', '.784'),
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Probabilistic Neural Network', '.70
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Polynomial Support Vector Machine',
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Radial Basis Support Vector Machine
                 c('Kangwanariyakul, Nantasenamat et al.', '2010', 'Bayesian Neural Network', '.784'),
                 c('Kumari and Godara', '2011', 'RIPPER', '.811'),
                 c('Kumari and Godara', '2011', 'Decision Tree', '.791'),
                 c('Kumari and Godara', '2011', 'Artificial Neural Network', ',801'),
                 c('Kumari and Godara', '2011', 'Support Vector Machine', '.841'),
                 c('Soni, Ansari et al.', '2011', 'Weighted Associative Classifier', '.578'),
                 c('Soni, Ansari et al.', '2011', 'Classification-Association', '.583'),
                 c('Soni, Ansari et al.', '2011', 'Classification-Multiple ClassAssociation', '.536'),
                 c('Soni, Ansari et al.', '2011', 'Classification-Predictive Association', '.523'),
                 c('Abdullah and Rajalaxmi', '2012', 'Decision Tree', '.507'),
                 c('Abdullah and Rajalaxmi', '2012', 'Random Forest', '.633'),
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Neural Network', '.805'),
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'J4.8 Decision Tree', '.779'),
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Support Vector Machine', '.842'),
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Feature Selection with Neural Network',
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Feature Selection with Decision Tree',
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Feature Selection with Support Vector Ma
                 c('Rajeswari, Vaithiyanathan et al.', '2013', 'Neural Network', '.805'),
                 c('Lakshmi, Krishna et al.',  '2013', 'Support Vector Machine' ,   '.781'),
                 c('Lakshmi, Krishna et al.',  '2013', 'Decision Tree', '.847'),
                 c('Lakshmi, Krishna et al.',  '2013', 'K Nearest Neighbor' ,   '.840'),
                 c('Lakshmi, Krishna et al.',  '2013', 'K Mean' ,   '.803'),
                 c('Pandey, Pandey et al.', '2013', 'COBWEB', '.020'),
                 c('Pandey, Pandey et al.', '2013', 'EM', '.815'),
                 c('Pandey, Pandey et al.', '2013', 'Farthest First', '.736'),
                 c('Pandey, Pandey et al.', '2013', 'Make Density Based Clusters', '.815'),
                 c('Pandey, Pandey et al.', '2013', 'Simple K-Means', '.809')
)

colnames(studies) <- c('AUTHOR', 'YEAR', 'TECHNIQUE', 'ACCURACY')

colnames(studies) <- c('AUTHOR', 'YEAR', 'TECHNIQUE', 'ACCURACY')

study_summ <- rbind(c('Logistic Regression', '.855'),
                    c('Random Forest', '.724'),
                    c('Support Vector Machine', '.809'),
                    c('Naive Bayes', '.819')
)

colnames(study_summ) <- c('TECHNIQUE', 'MEDIAN ACCURACY')


# DATA EXPLORATION <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

# Load data

data <- read.csv ('https://raw.githubusercontent.com/betsyrosalen/DATA_621_Business_Analyt_and_Data_Mini
                   stringsAsFactors = F, header = T)

data$target <- as.factor(data$target)

vars <- rbind(c('age','Age','continuous numerical predictor'),
              c('sex','Sex. Female = 0, Male = 1 ','categorical predictor'),
              c('cp','Chest pain type. Scale of 0 to 4','categorical predictor'),
              c('trestbps','Diastolic blood pressure in mmHg','continuous numerical predictor'),
              c('chol','Serum cholesterol (mg/dl)','continuous numerical predictor'),
              c('fbs','Fasting blood sugar. Greater than 120mg/dl, value of 0 or 1','categorical predic
              c('restecg','Resting ECG. Value of 0, 1, or 2','categorical predictor'),
              c('thalach','Maximum heartrate achieved from thallium test','continuous numerical predicto
              c('exang','Exercise-induced angina. Value of 0 or 1','categorical predictor'),
              c('oldpeak','Old-peak.ST depression induced by exercise relative to rest','continuous nume
              c('slope','Slope of peak exercise ST segment, value of 1, 2, or 3','categorical predictor
              c('ca','Number of major vessels (0-3) colored by fluoroscopy','categorical predictor'),
              c('thal','Exercise thallium scintigraphic defects','categorical predictor'),
              c('target','Response Variable - No Heart Disease = 0, Heart Disease = 1','categorical pre

colnames(vars) <- c('VARIABLE','DEFINITION','TYPE')
```

```
# Bootstrap surrogate data using synthpop

# https://cran.r-project.org/web/packages/synthpop/vignettes/synthpop.pdf
# https://cran.r-project.org/web/packages/synthpop/synthpop.pdf
# https://www.r-bloggers.com/generating-synthetic-data-sets-with-synthpop-in-r/
# https://www.geos.ed.ac.uk/homes/graab/synthpop.pdf

syn_obj <- synthpop::syn(data = data, m = 20)
# creates 20 synthetic datasets based on original dataset and its variables' distributions
syn_dflist <- syn_obj$syn  # extract list of synthesized data frames from synds object
syn_df <- dplyr::bind_rows(syn_dflist, .id = 'column_label')
# unlist synds object and configures 6,060 synthetic observations as df


# Summary Statistics

orig_data <- data
syn_data <-  syn_df
syn_data$column_label <- NULL
colnames(orig_data)[colnames(orig_data)=="?..age"] <- "age"
colnames(syn_data)[colnames(syn_data)=="?..age"] <- "age"

orig_data$sex <- as.factor(orig_data$sex)
orig_data$cp <- as.factor(orig_data$cp)
orig_data$ca <- as.factor(orig_data$ca)
orig_data$fbs <- as.factor(orig_data$fbs)
orig_data$restecg <- as.factor(orig_data$restecg)
orig_data$slope <- as.factor(orig_data$slope)
orig_data$target <- as.factor(orig_data$target)
orig_data$thal <- as.factor(orig_data$thal)
orig_data$exang <- as.factor(orig_data$exang)

syn_data$sex <- as.factor(syn_data$sex)
syn_data$cp <- as.factor(syn_data$cp)
syn_data$ca <- as.factor(syn_data$ca)
syn_data$fbs <- as.factor(syn_data$fbs)
syn_data$restecg <- as.factor(syn_data$restecg)
syn_data$slope <- as.factor(syn_data$slope)
syn_data$target <- as.factor(syn_data$target)
syn_data$thal <- as.factor(syn_data$thal)
syn_data$exang <- as.factor(syn_data$exang)


data_num <- orig_data[, c( 'age','trestbps', 'chol', 'thalach', 'oldpeak')]
data_cat <- orig_data[, c('sex', 'cp','ca', 'exang','fbs', 'restecg', 'slope',
                          'target', 'thal' )]
syn_num <- syn_data[, c( 'age','trestbps', 'chol','thalach', 'oldpeak')]
syn_cat <- syn_data[, c('sex', 'cp','ca', 'exang', 'fbs', 'restecg', 'slope',
                        'target', 'thal')]

### Summary Statistics
orig_num_stats <- describe(data_num)[,c(2,8,3,5,9,4)]
syn_num_stats <- describe(syn_num)[,c(2,8,3,5,9,4)]
```

```
orig_cat_stats <- summary(data_cat)
syn_cat_stats <- summary(syn_cat)
# orig_cat_stats <- summary(data_cat[, c( 'cp','ca',  'restecg', 'slope', 'thal')])
# syn_cat_stats <- summary(syn_cat[, c('cp','ca',   'restecg', 'slope', 'thal')])
# orig_cat_stats_b <- summary(data_cat[, c('exang', 'fbs', 'sex', 'target')])
# syn_cat_stats_b <- summary(syn_cat[, c('exang', 'fbs', 'sex', 'target')])


# Outliers

# Data Distribution

data_num_hist <- orig_data[, c( 'age','trestbps', 'chol', 'thalach', 'oldpeak',
                                'target')]
hist.num <- data_num_hist %>%
  gather(-target, key = "var", value = "val") %>%
  ggplot(aes(x = val, fill=factor(target))) +
  geom_histogram(position="dodge", bins=10, alpha=0.5) +
  facet_wrap(~ var, scales = "free") +
  scale_fill_manual("target",values = c("#58BFFF", "#3300FF")) +
  xlab("") +
  ylab("") +
  theme(panel.background = element_blank(), legend.position="top")

bar.cat <- data_cat %>%
  gather(-target, key = "var", value = "val") %>%
  ggplot(aes(x = val, fill=factor(target))) +
  geom_bar(position="dodge", alpha=0.5) +
  facet_wrap(~ var, scales = "free") +
  scale_fill_manual("target",values = c("#58BFFF", "#3300FF")) +
  xlab("") +
  ylab("") +
  theme(panel.background = element_blank(), legend.position="top") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))


# Scaled BoxPlots

scaled.train.num <- as.data.table(scale(orig_data[, c( 'age','trestbps', 'chol',
                                                       'thalach', 'oldpeak')]))
melt.train <- melt(scaled.train.num)

scaled.boxplots <- ggplot(melt.train, aes(variable, value)) +
  geom_boxplot(width=.5, fill="#58BFFF", outlier.colour="red", outlier.size = 1) +
  stat_summary(aes(colour="mean"), fun.y=mean, geom="point",
               size=2, show.legend=TRUE) +
  stat_summary(aes(colour="median"), fun.y=median, geom="point",
               size=2, show.legend=TRUE) +
  coord_flip() +
  #scale_y_continuous(labels = scales::comma,
  #                   breaks = seq(0, 110, by = 10)) +
  labs(colour="Statistics", x="", y="") +
  scale_colour_manual(values=c("#9900FF", "#3300FF")) +
  theme(panel.background=element_blank(), legend.position="top")
```

```
# Linear relationship between each numeric predictor and the target

linear_graph_data <- orig_data[, c('age','trestbps', 'chol', 'thalach', 'oldpeak',
                                    'target')]
boxplots.target <- linear_graph_data  %>%
  gather(-target,key = "var", value = "val") %>%
  ggplot(aes(x=factor(target), y=val)) +
  geom_boxplot(width=.5, fill="#58BFFF", outlier.colour="red", outlier.size = 1) +
  stat_summary(aes(colour="mean"), fun.y=mean, geom="point",
               size=2, show.legend=TRUE) +
  stat_summary(aes(colour="median"), fun.y=median, geom="point",
               size=2, show.legend=TRUE) +
  facet_wrap(~ var, scales = "free", ncol=5) +
  labs(colour="Statistics", x="", y="") +
  scale_colour_manual(values=c("#9900FF", "#3300FF")) +
  theme(panel.background=element_blank())


## Correlation

plot.data <- linear_graph_data
plot.data$target<- factor(plot.data$target)
corr.plot2 <- plot.data %>%
  ggscatmat(color="target", alpha=0.1) +
  scale_color_manual(values=c("#58BFFF", "#3300FF")) +
  theme(panel.background=element_blank(), legend.position="top",
        axis.text.x = element_text(angle=-40, vjust=1, hjust=0))


## Heart disease by gender
#gender_data <-  orig_data
#levels(gender_data$target) = c("No disease","Disease")
#levels(gender_data$sex) = c("female","male","")
#gender_hist <-  mosaicplot(gender_data$sex ~ gender_data$target,
                           #main="",shade=FALSE,color = c("#58BFFF", "#3300FF"),
                           #xlab="Gender", ylab="Heart disease")

## Heart disease by Fasting blood sugar
#levels(gender_data$target) = c("No disease","Disease")
#levels(gender_data$fbs) = c("0","1","")
#fbs_hist <-  mosaicplot(gender_data$fbs ~ gender_data$target,
                        #main="",shade=FALSE,color = c("#58BFFF", "#3300FF"),
                        #xlab="fbs - Fasting blood sugar", ylab="Heart disease")

## Heart disease by Fasting blood sugar
#levels(gender_data$target) = c("No disease","Disease")
#levels(gender_data$exang) = c("0","1","")
#exang_hist <-  mosaicplot(gender_data$exang ~ gender_data$target,
                          #main="",shade=FALSE,color = c("#58BFFF", "#3300FF"),
                          #xlab="exang - Exercise-induced angina ", ylab="Heart disease")

#bar_cp <- ggplot(orig_data,aes(factor(cp)))+geom_bar(aes(fill = target), position = "dodge")
```

```
# Compare distribution of original data and synthesized data
compare_synthoriginal <- compare(syn_obj, data)
# visually compare of synthetic datasets vs original data
# syn_csv <- write.syn(syn_obj, 'csv')

# style_p <- function(p){
#   for (plot in p){plot +
#     scale_fill_manual("target",values = c("#58BFFF", "#3300FF")) +
#     theme(panel.background = element_blank(), legend.position="top") +
#     theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
# }}
#
# lapply(compare_synthoriginal$plots, style_p)

compare_synthoriginal$plots[[1]] <- compare_synthoriginal$plots[[1]]+
  scale_fill_manual(values = c("#58BFFF", "#3300FF")) +
  theme(panel.background = element_blank(), legend.position="top") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

compare_synthoriginal$plots[[2]] <- compare_synthoriginal$plots[[2]]+
  scale_fill_manual(values = c("#58BFFF", "#3300FF")) +
  theme(panel.background = element_blank(), legend.position="top") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

compare_synthoriginal$plots[[3]] <- compare_synthoriginal$plots[[3]]+
  scale_fill_manual(values = c("#58BFFF", "#3300FF")) +
  theme(panel.background = element_blank(), legend.position="top") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

compare_synthoriginal$plots[[4]] <- compare_synthoriginal$plots[[4]]+
  scale_fill_manual(values = c("#58BFFF", "#3300FF")) +
  theme(panel.background = element_blank(), legend.position="top") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

# [Betsy / Gabby, please remove the folllowing code block if no one is using]
# Split data into test and train using cross-validation
# folds <- 10  # set to 10 provisionally
# train_bootstrap <- caret::trainControl(method = 'boot', number = folds)
# bootstrap resampling approach
# train_kfold <- caret::trainControl(method = 'cv', number = folds)
# k-fold cross validation
# train_kfoldrpt <- caret::trainControl(method = 'repeatedcv', number = folds, repeats = 3)
# k-fold cross validation, provisionally set to 3 repeats but explore setting
# train_loocv <- caret::trainControl(method = 'LOOCV')
# syn_df$column_label <- NULL


#Missing Data

# DATA PREPARATION <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

##### Support vector machines model >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
# cv: https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/
# caret vignette: https://cran.r-project.org/web/packages/caret/vignettes/caret.html
# caret overview: http://www.rebeccabarter.com/blog/2017-11-17-caret_tutorial/
# SVM setup: https://topepo.github.io/caret/train-models-by-tag.html#Support_Vector_Machines
# SVM setup: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
# SVM setup: http://dataaspirant.com/2017/01/19/support-vector-machine-classifier-implementation-r-care
# SVM maths: http://web.mit.edu/6.034/wwwbob/svm.pdf
# SVM tuning: https://blog.revolutionanalytics.com/2015/10/the-5th-tribe-support-vector-machines-and-ca


# Create test and train datasets
# [UPDATE FOR CONSISTENCY WITH ROSE / LYDIA]
train_flag_orig <- caret::createDataPartition(orig_data$target,
                                              p = .75,
                                              list = FALSE)
svm_train_orig <- orig_data[train_flag_orig,]
svm_test_orig <- orig_data[-train_flag_orig,]
svm_train_orig$target <- as.factor(make.names(svm_train_orig$target))
# include make.names() so 0-1 coded target variable is syntactically valid for train()
svm_test_orig$target <- as.factor(make.names(svm_test_orig$target))
# include make.names() so 0-1 coded target variable is syntactically valid for train()

train_flag_syn <- caret::createDataPartition(syn_data$target,
                                             p = .75,
                                             list = FALSE)
svm_train_syn <- syn_data[train_flag_syn,]
svm_test_syn <- syn_data[-train_flag_syn,]
svm_train_syn$target <- as.factor(make.names(svm_train_syn$target))
# include make.names() so 0-1 coded target variable is syntactically valid for train()
svm_test_syn$target <- as.factor(make.names(svm_test_syn$target))
# include make.names() so 0-1 coded target variable is syntactically valid for train()


# Set up cross-validation methods

# Cross-validation generic setup
ctrl_svm1 <- caret::trainControl(method = 'repeatedcv',  # resampling method is repeated cross-validati
                                 number = 10,  # 10 resampling iterations
                                 repeats = 5) #,  # conduct 5 repetitions of cross-validation

# Cross-validation for use with ROC metric argument
ctrl_svm2 <- caret::trainControl(method = 'repeatedcv',  # resampling method is repeated cross-validati
                                 repeats = 5,  # conduct 5 repetitions of cross-validation
                                 summaryFunction = twoClassSummary,
                                 classProbs = TRUE)


# First SVM RBF model (original data)

# Build first SVM RBF model with original data
mod_svmradial1 <- readRDS("./source/model/mod_svmradial1.rds")
# mod_svmradial1 <- caret::train(target ~ .,
#                                data = svm_train_orig,
#                                 method = 'svmRadial',  # RBF model
#                                 trControl = ctrl_svm2,  # cross-validation for ROC
#                                 preProcess = c('center', 'scale'),  # preprocess data to center and sca
#                                 metric = 'ROC')  # ROC-based evaluation
# saveRDS(mod_svmradial1, "./source/model/mod_svmradial1.rds")

# Predict values based on first SVM RBF model
testpred_svmradial1 <- predict(mod_svmradial1,
                               newdata = svm_test_orig)

# Create confusion matrix for first SVM RBF model
confmtrx_svmradial1 <- confusionMatrix(testpred_svmradial1,
                                       svm_test_orig$target)

# Plot ROC curve for first SVM RBF model
plot_svmradial1 <- ggplot(mod_svmradial1)


# Second SVM RBF model (original data)

# Create grid search of tuning parameters for second SVM RBF model
tunegrid_svmradial2 <- expand.grid(sigma = c(.01, .02, .03, .04, .05),
                             C = c(0.25, 0.4, .5, .6, .75))

# Build second, tuned SVM RBF model with original data
mod_svmradial2 <- readRDS("./source/model/mod_svmradial2.rds")
# mod_svmradial2 <- caret::train(target ~ .,
#                                 data = svm_train_orig,
#                                 method = 'svmRadial',  # RBF model
#                                 trControl = ctrl_svm2,  # cross-validation for ROC
#                                 preProcess = c('center', 'scale'),  # preprocess data to center and sca
#                                 tuneGrid = tunegrid_svmradial2,  # grid search to tune on C and sigma
#                                 metric = 'ROC')  # ROC-based evaluation
# saveRDS(mod_svmradial2, "./source/model/mod_svmradial2.rds")

# Predict values based on second, tuned SVM RBF model
testpred_svmradial2 <- predict(mod_svmradial2,
                               newdata = svm_test_orig)

# Create confusion matrix for second, tuned SVM RBF model
confmtrx_svmradial2 <- confusionMatrix(testpred_svmradial2,
                                       svm_test_orig$target)

# Plot ROC curve for second, tuned SVM RBF model
plot_svmradial2 <- ggplot(mod_svmradial2)


# Third SVM RBF model (synthetic data)

# Build third SVM RBF model with synthesized data
mod_svmradial3 <- readRDS("./source/model/mod_svmradial3.rds")
# mod_svmradial3 <- caret::train(target ~ .,
#                                 data = svm_train_syn,
#                                 method = 'svmRadial',  # RBF model
#                                trControl = ctrl_svm2,  # cross-validation for ROC
#                                preProcess = c('center', 'scale'),  # preprocess data to center and sca
#                                metric = 'ROC')  # ROC-based evaluation
# saveRDS(mod_svmradial3, "./source/model/mod_svmradial3.rds")

# Predict values based on third SVM RBF model with synthesized data
testpred_svmradial3 <- predict(mod_svmradial3,
                               newdata = svm_test_syn)

# Create confusion matrix for third SVM RBF model with synthesized data
confmtrx_svmradial3 <- confusionMatrix(testpred_svmradial3,
                                       svm_test_syn$target)

# Plot ROC curve for third SVM RBF model with synthesized data
plot_svmradial3 <- ggplot(mod_svmradial3)


# SVM Linear model

# Use grid search of tuning parameters for second SVM RBF model
tunegrid_svmlinear1 <- expand.grid(C = c(0.01, 0.05, 0.1, 0.25,
                                         0.5, 0.75, 1, 1.25, 1.5,
                                         1.75, 2, 5))

# Build first SVM linear model with original data
mod_svmlinear1 <- readRDS("./source/model/mod_svmlinear1.rds")
# mod_svmlinear1 <- caret::train(target ~ .,
#                                data = svm_train_syn,
#                                method = 'svmLinear',  # check and confirm, alternate = 'pls'
#                                trControl = ctrl_svm1,
#                                preProcess = c('center', 'scale'),  # check and confirm
#                                tuneGrid = tunegrid_svmlinear1,
#                                tunelength = 10) # check and confirm
# saveRDS(mod_svmlinear1, "./source/model/mod_svmlinear1.rds")

# Predict values based on linear SVM RBF model
testpred_svmlinear1 <- predict(mod_svmlinear1,
                               newdata = svm_test_syn)

# Create confusion matrix for first SVM linear model
confmtrx_svmlinear1 <- confusionMatrix(testpred_svmlinear1,
                                       svm_test_syn$target)

# Plot ROC curve for first SVM linear model with synthesized data
plot_svmlinear1 <- ggplot(mod_svmlinear1)


#================================================================================#

## Model 5 - Naive Bayes Model (see notes)

#train data
orig_traindata_nb <- svm_train_orig
colnames(orig_traindata_nb)[colnames(orig_traindata_nb)=="ï..age"] <- "age"
```

```
syn_traindata_nb <- svm_train_syn
colnames(syn_traindata_nb)[colnames(syn_traindata_nb)=="ï..age"] <- "age"

#test data
orig_testdata_nb <- svm_test_orig
colnames(orig_testdata_nb)[colnames(orig_testdata_nb)=="ï..age"] <- "age"

syn_testdata_nb <- svm_test_syn
colnames(syn_testdata_nb)[colnames(syn_testdata_nb)=="ï..age"] <- "age"


#chol as category
nb.cat <- function(x, lower = 100, upper, by = 80,sep = "-", above.char = "+")
      {labs <- c(paste(seq(lower, upper - by, by = by),
      seq(lower + by - 1, upper - 1, by = by),sep = sep),paste(upper, above.char, sep = ""))
cut(floor(x), breaks = c(seq(lower, upper, by = by), Inf), right = FALSE, labels = labs)}

orig_traindata_nb$cholGroup <-  nb.cat(orig_traindata_nb$chol, upper = 350)
syn_traindata_nb$cholGroup <-  nb.cat(syn_traindata_nb$chol, upper = 350)

orig_testdata_nb$cholGroup <-  nb.cat(orig_testdata_nb$chol, upper = 350)
syn_testdata_nb$cholGroup <-  nb.cat(syn_testdata_nb$chol, upper = 350)

orig_traindata_nb$chol <- NULL
syn_traindata_nb$chol <-  NULL

orig_testdata_nb$chol <-  NULL
syn_testdata_nb$chol <-  NULL


#remove age and sex

orig_traindata_nb$age <-  NULL
syn_traindata_nb$age <-  NULL
orig_testdata_nb$age <-  NULL
syn_testdata_nb$age <-  NULL

orig_traindata_nb$sex <-  NULL
syn_traindata_nb$sex <-  NULL
orig_testdata_nb$sex <-  NULL
syn_testdata_nb$sex <-  NULL

model_orig_nb <- e1071::naiveBayes(target ~ ., data = orig_traindata_nb)
orig_testdata_nb$pred <- predict(model_orig_nb, orig_testdata_nb)
confusionMatrix_orig_nb <- caret::confusionMatrix(orig_testdata_nb$pred ,
                                                  orig_testdata_nb$target)

model_syn_nb <- e1071::naiveBayes(target ~ ., data = syn_traindata_nb)
syn_testdata_nb$pred <- predict(model_syn_nb, syn_testdata_nb)
confusionMatrix_syn_nb <- caret::confusionMatrix(syn_testdata_nb$pred,
                                                  syn_testdata_nb$target)
```

```
##### LOGISTIC MODELS >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

set.seed(123)
# Create test and train datasets
lr_train_flag_orig <- caret::createDataPartition(orig_data$target,
                                              p = .75,
                                              list = FALSE)
lr_svm_train_orig <- orig_data[train_flag_orig,]
lr_svm_test_orig <- orig_data[-train_flag_orig,]

lr_train_flag_syn <- caret::createDataPartition(syn_data$target,
                                             p = .75,
                                             list = FALSE)
lr_svm_train_syn <- syn_data[lr_train_flag_syn,]
lr_svm_test_syn <- syn_data[-lr_train_flag_syn,]

#############
# ORIG_DATA #
#############

# Summary of variables for orig_data
# 14:48:01> summary(lm(as.numeric(target)~., orig_data))
#
# Call:
#   lm(formula = as.numeric(target) ~ ., data = orig_data)
#
# Residuals:
#   Min       1Q   Median       3Q      Max
# -1.01468 -0.18519  0.03069  0.22575  1.02624
#
# Coefficients:
#   Estimate Std. Error t value Pr(>|t|)
# (Intercept)  1.4632832  0.3688720   3.967 9.26e-05 ***
#   age          0.0027129  0.0026488   1.024 0.306638
# sex1        -0.1640717  0.0482271  -3.402 0.000766 ***
#   cp1          0.1688621  0.0639741   2.640 0.008767 **
#   cp2          0.2252316  0.0537885   4.187 3.78e-05 ***
#   cp3          0.2657100  0.0811006   3.276 0.001184 **
#   trestbps    -0.0023401  0.0012187  -1.920 0.055844 .
# chol        -0.0003189  0.0004021  -0.793 0.428416
# fbs1         0.0333463  0.0574909   0.580 0.562362
# restecg1     0.0469891  0.0408340   1.151 0.250823
# restecg2    -0.0870584  0.1757245  -0.495 0.620689
# thalach      0.0018708  0.0011189   1.672 0.095634 .
# exang1      -0.0935220  0.0499945  -1.871 0.062437 .
# oldpeak     -0.0415842  0.0230281  -1.806 0.072023 .
# slope1      -0.0653951  0.0859299  -0.761 0.447281
# slope2       0.0736023  0.0950718   0.774 0.439480
# ca1         -0.2703534  0.0530757  -5.094 6.46e-07 ***
#   ca2         -0.3426057  0.0682472  -5.020 9.20e-07 ***
#   ca3         -0.2960027  0.0861537  -3.436 0.000680 ***
#   ca4          0.0443264  0.1555910   0.285 0.775939
# thal1        0.2283726  0.2543715   0.898 0.370068
# thal2        0.2879971  0.2418275   1.191 0.234694
# thal3        0.0784919  0.2436939   0.322 0.747623
# ---
#   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.335 on 280 degrees of freedom
# Multiple R-squared:  0.5818,  Adjusted R-squared:  0.549
# F-statistic: 17.71 on 22 and 280 DF,  p-value: < 2.2e-16

# 14:52:43> summ(lm(as.numeric(target) ~ sex + ca + cp, lr_svm_train_orig))
# MODEL FIT:
# F(11,231) = 16.20, p = 0.00
# R2 = 0.43
# Adj. R2 = 0.41
# AIC = 235.6967  BIC = 270.6273
lm_select1_orig <- lm(as.numeric(target) ~ sex + ca + cp, lr_svm_train_orig)
lm_select1_pred_orig <- as.factor(round(predict(lm_select1_orig, lr_svm_test_orig))-1)
lm_select1_pred_orig_confusion <- confusionMatrix(lm_select1_pred_orig,
                                                reference=lr_svm_test_orig$target)
lm_select1_pred_origXsyn <- as.factor(round(predict(lm_select1_orig, lr_svm_test_syn))-1)
lm_select1_pred_origXsyn_confusion <- confusionMatrix(lm_select1_pred_origXsyn,
                                                    reference=lr_svm_test_syn$target)

# 15:34:44> summ(lm(as.numeric(target) ~ sex * ca + cp, lr_svm_train_orig))
# MODEL FIT:
# F(8,234) = 21.78, p = 0.00
# R2 = 0.44
# Adj. R2 = 0.41
# AIC = 237.9943  BIC = 283.4041
lm_select2_orig <- lm(as.numeric(target) ~ sex * ca + cp, lr_svm_train_orig)
lm_select2_pred_orig <- as.factor(round(predict(lm_select2_orig, lr_svm_test_orig))-1)
lm_select2_pred_orig_confusion <- confusionMatrix(lm_select2_pred_orig,
                                                reference=lr_svm_test_orig$target)
lm_select2_pred_origXsyn <- as.factor(round(predict(lm_select2_orig, lr_svm_test_syn))-1)
lm_select2_pred_origXsyn_confusion <- confusionMatrix(lm_select2_pred_origXsyn,
                                                    reference=lr_svm_test_syn$target)

# 15:36:36> summ(lm(as.numeric(target)~., lr_svm_train_orig[sapply(lr_svm_train_orig, is.numeric)|names
# MODEL FIT:
# F(5,237) = 18.98, p = 0.00
# R2 = 0.29
# Adj. R2 = 0.27
# AIC = 283.0819  BIC = 307.5334
lm_numeric_orig <- lm(as.numeric(target)~., lr_svm_train_orig[sapply(lr_svm_train_orig, is.numeric)|name
#lm_numeric_pred_origXsyn <- as.factor(round(predict(lm_numeric_orig, lr_svm_test_syn))-1)
lm_numeric_pred_orig <- as.factor(sapply(round(predict(lm_numeric_orig, lr_svm_test_orig))-1,
                                         function (x) ifelse(x < 0, 0, ifelse(x>1, 1, x))))
                                         # if there is an error about more levels, this is where you f
lm_numeric_pred_orig_confusion <- confusionMatrix(lm_numeric_pred_orig,
                                                reference=lr_svm_test_orig$target)
lm_numeric_pred_origXsyn <- as.factor(sapply(round(predict(lm_numeric_orig, lr_svm_test_syn))-1,
                                             function (x) ifelse(x < 0, 0, ifelse(x>1, 1, x))))
lm_numeric_pred_origXsyn_confusion <- confusionMatrix(lm_numeric_pred_origXsyn,
                                                    reference=lr_svm_test_syn$target)
```

```
# 15:38:03> summ(lm(as.numeric(target)~., lr_svm_train_orig[!sapply(lr_svm_train_orig, is.numeric)|names
# MODEL FIT:
# F(17,225) = 17.73, p = 0.00
# R2 = 0.57
# Adj. R2 = 0.54
# AIC = 182.3636  BIC = 248.7317
lm_factors_orig <- lm(as.numeric(target)~., lr_svm_train_orig[!sapply(lr_svm_train_orig, is.numeric)|nam
lm_factors_pred_orig <- as.factor(round(predict(lm_factors_orig, lr_svm_test_orig))-1)
lm_factors_pred_orig_confusion <- confusionMatrix(lm_factors_pred_orig,
                                                  reference=lr_svm_test_orig$target)
lm_factors_pred_orig_confusion2 <- confusionMatrix(lm_factors_pred_orig,
                                                    reference=lr_svm_test_orig$target,
                                                    mode = "prec_recall")
lm_factors_pred_origXsyn <- as.factor(round(predict(lm_factors_orig, lr_svm_test_syn))-1)
lm_factors_pred_origXsyn_confusion <- confusionMatrix(lm_factors_pred_origXsyn,
                                                      reference=lr_svm_test_syn$target)


############
# SYN_DATA #
############

# Summary of variables for syn_data
# 15:40:07> summary(lm(as.numeric(target)~., syn_data))
# Call:
#   lm(formula = as.numeric(target) ~ ., data = syn_data)
#
# Residuals:
#   Min       1Q   Median       3Q      Max
# -1.10137 -0.26218  0.01789  0.25941  1.16579
#
# Coefficients:
#   Estimate Std. Error t value Pr(>|t|)
# (Intercept)  2.017e+00  9.072e-02  22.234  < 2e-16 ***
#   age         -6.828e-03  6.410e-04 -10.652  < 2e-16 ***
#   sex1        -1.412e-01  1.184e-02 -11.924  < 2e-16 ***
#   cp1          3.300e-01  1.563e-02  21.120  < 2e-16 ***
#   cp2          3.121e-01  1.313e-02  23.774  < 2e-16 ***
#   cp3          2.801e-01  2.033e-02  13.776  < 2e-16 ***
#   trestbps    -6.030e-04  3.017e-04  -1.999 0.045686 *
#   chol        -3.261e-04  9.885e-05  -3.299 0.000976 ***
#   fbs1         1.844e-02  1.373e-02   1.343 0.179344
# restecg1     1.844e-02  1.020e-02   1.808 0.070616 .
# restecg2     3.291e-02  4.748e-02   0.693 0.488239
# thalach      7.416e-04  2.721e-04   2.726 0.006435 **
#   exang1      -1.792e-02  1.228e-02  -1.459 0.144678
# oldpeak     -6.553e-03  5.560e-03  -1.179 0.238598
# slope1       4.967e-02  1.924e-02   2.582 0.009851 **
#   slope2       6.780e-02  2.203e-02   3.077 0.002099 **
#   ca1         -2.220e-01  1.266e-02 -17.534  < 2e-16 ***
#   ca2         -2.460e-01  1.603e-02 -15.341  < 2e-16 ***
#   ca3         -2.615e-01  2.147e-02 -12.176  < 2e-16 ***
#   ca4         -2.183e-01  3.605e-02  -6.057 1.47e-09 ***
#   thal1       -3.547e-02  5.881e-02  -0.603 0.546399
# thal2       -1.230e-02  5.590e-02  -0.220 0.825896
# thal3       -2.212e-01  5.622e-02  -3.935 8.42e-05 ***
#   ---
#   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.3745 on 6037 degrees of freedom
# Multiple R-squared:  0.4397,  Adjusted R-squared:  0.4377
# F-statistic: 215.4 on 22 and 6037 DF,  p-value: < 2.2e-16

# 15:40:23> summ(lm(as.numeric(target) ~ age + sex + chol + ca + cp, lr_svm_train_syn))
# MODEL FIT:
# F(10,4837) = 317.00, p = 0.00
# R2 = 0.40
# Adj. R2 = 0.39
# AIC = 4605.077  BIC = 4682.913
lm_select_syn <- lm(as.numeric(target) ~ age + sex + chol + ca + cp, lr_svm_train_syn)
lm_select_pred_syn <- as.factor(round(predict(lm_select_syn, lr_svm_test_syn))-1)
lm_select_pred_syn_confusion <- confusionMatrix(lm_select_pred_syn,
                                                reference=lr_svm_test_syn$target)
lm_select_pred_synXorig <- as.factor(round(predict(lm_select_syn, lr_svm_test_orig))-1)
lm_select_pred_synXorig_confusion <- confusionMatrix(lm_select_pred_synXorig,
                                                      reference=lr_svm_test_orig$target)

# summ(lm(as.numeric(target)~., lr_svm_train_syn[sapply(lr_svm_train_syn, is.numeric)|
                                                  names(lr_svm_train_syn)=="target"]))
# MODEL FIT:
# F(5,4842) = 209.21, p = 0.00
# R2 = 0.18
# Adj. R2 = 0.18
# AIC = 6090.317  BIC = 6135.722
lm_numeric_syn <- lm(as.numeric(target)~., lr_svm_train_syn[sapply(lr_svm_train_syn, is.numeric)|names(
lm_numeric_pred_syn <- as.factor(round(predict(lm_numeric_syn, lr_svm_test_syn))-1)
lm_numeric_pred_syn_confusion <- confusionMatrix(lm_numeric_pred_syn,
                                                  reference=lr_svm_test_syn$target)
lm_numeric_pred_synXorig <- as.factor(round(predict(lm_numeric_syn, lr_svm_test_orig))-1)
lm_numeric_pred_synXorig_confusion <- confusionMatrix(lm_numeric_pred_synXorig,
                                                      reference=lr_svm_test_orig$target)

# 15:47:08> summ(lm(as.numeric(target)~., lr_svm_train_syn[!sapply(lr_svm_train_syn, is.numeric)|names(
# MODEL FIT:
# F(17,4830) = 206.19, p = 0.00
# R2 = 0.42
# Adj. R2 = 0.42
# AIC = 4417.274  BIC = 4540.515
lm_factors_syn <- lm(as.numeric(target)~., lr_svm_train_syn[!sapply(lr_svm_train_syn, is.numeric)|names
lm_factors_pred_syn <- as.factor(round(predict(lm_factors_syn, lr_svm_test_syn))-1)
lm_factors_pred_syn_confusion <- confusionMatrix(lm_factors_pred_syn,
                                                  reference=lr_svm_test_syn$target)
lm_factors_pred_syn_confusion2 <- confusionMatrix(lm_factors_pred_syn,
                                                  reference=lr_svm_test_syn$target,
                                                  mode = "prec_recall")
lm_factors_pred_synXorig <- as.factor(round(predict(lm_factors_syn, lr_svm_test_orig))-1)
lm_factors_pred_synXorig_confusion <- confusionMatrix(lm_factors_pred_synXorig,
                                                      reference=lr_svm_test_orig$target)
```

```
lr_compared <- data.frame(t(lm_select1_pred_orig_confusion$overall))
lr_compared <- rbind(lr_compared, data.frame(t(lm_select1_pred_origXsyn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_select2_pred_orig_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_select2_pred_origXsyn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_numeric_pred_orig_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_numeric_pred_origXsyn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_factors_pred_orig_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_factors_pred_origXsyn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_select_pred_syn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_select_pred_synXorig_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_numeric_pred_syn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_numeric_pred_synXorig_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_factors_pred_syn_confusion$overall)))
lr_compared <- rbind(lr_compared, data.frame(t(lm_factors_pred_synXorig_confusion$overall)))
rownames(lr_compared) <- c("select1_o", "select1_oXs", "select2_o", "select2_oXs",
                           "numeric_o", "numeric_oXs", "factors_o", "factors_oXs",
                           "select_s", "select_sXo", "numeric_s", "numeric_sXo",
                           "factors_s", "factors_sXo")
lr_compared <- lr_compared[order(-lr_compared$Accuracy),]


##############
# BEST MODEL #
##############

lr_plots <- autoplot(lm_factors_orig, which = 1:6, colour = "#58BFFF",
                                 smooth.colour = 'red', smooth.linetype = 'solid',
                                 ad.colour = 'black',
                                 label.size = 3, label.n = 5, label.colour = "#3300FF",
                                 ncol = 2) +
  theme(panel.background=element_blank())


##### Random Forest Models >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

registerDoMC(cores=8)
# Random Forest <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
# seed <- 23
# # set predictor variable x, target variable y
# x <- orig_data[,1:13]
# y <- orig_data[,14]
#

# Baseline model
rf.baseline <- readRDS("./source/model/rf_base.rds")
# rf.baseline <- train(target~.,
#                       data=orig_data,
#                       method="rf",
#                       metric="Accuracy",
#                       tuneGrid=expand.grid(.mtry=sqrt(ncol(x))),
#                       trControl=trainControl(method="repeatedcv", number=10, repeats=3))

#--------------------------------------------------------------------------------
```

```
# Use syn_data as in-sample, orig_data as out-of-sample with cross validation using tuned hyper paramet
rf.syn.clf <- readRDS("./source/model/rf_syn_clf.rds")
rf.syn.clf.cv <- readRDS("./source/model/rf_syn_clf_cv.rds")

# set.seed(seed)
# split <- sample.split(syn_data$target, SplitRatio = 0.75)
# training_set <- subset(syn_data, split == TRUE)
# test_set <- subset(syn_data, split == FALSE)
#
# ## Fitting classifier to the Training set
# rf.syn.clf <- randomForest(x = training_set[-14],
#                            y = training_set$target,
#                            ntree = 1000, # Insert tuned hyperparameters
#                            mtry = 1)
# #saveRDS(rf.syn.clf, "./model/rf_syn_clf.rds")
# ## cross validation
# rf.syn.clf.cv <- rf.crossValidation(rf.syn.clf, training_set[-14], p=0.10, n=100, ntree=1000)
#saveRDS(rf.syn.clf.cv, "./model/rf_syn_clf_cv.rds")

## Predicting the Test set results
syn.y.pred <- predict(rf.syn.clf, newdata = orig_data[-14])
syn.cm <- table(orig_data$target, syn.y.pred)

#-------------------------------------------------------------------------------------

# Use orig_data as in-sample and out-of-sample after split with cross validation using tuned hyper para
rf.org.clf <- readRDS("./source/model/rf_org_clf.rds")
rf.org.clf.cv <- readRDS("./source/model/rf_org_clf_cv.rds")
#
# ## split data
# set.seed(seed)
# split <- sample.split(orig_data$target, SplitRatio = 0.75)
# training_set <- subset(orig_data, split == TRUE)
# test_set <- subset(orig_data, split == FALSE)
#
# ## Fitting classifier to the Training set
# rf.org.clf = randomForest(x = training_set[-14],
#                            y = training_set$target,
#                            ntree = 1000, # Insert tuned hyperparameters
#                            mtry = 1)
# saveRDS(rf.org.clf, "./model/rf_org_clf.rds")
# rf.org.clf.cv <- rf.crossValidation(rf.org.clf, training_set[-14], p=0.10, n=100, ntree=1000)
# saveRDS(rf.org.clf.cv, "./model/rf_org_clf_cv.rds")
## Predicting the Test set results
org.y.pred <- predict(rf.org.clf, newdata = orig_data[-14])
org.cm <- table(orig_data$target, org.y.pred)

#-------------------------------------------------------------------------------------

# Confusion matrix of prediction
rf.org.cm <- caret::confusionMatrix(table(predicted=as.numeric(levels(org.y.pred)[as.integer(org.y.pred
                                          actual = orig_data$target), mode = 'everything')
```

```
rf.syn.cm <- caret::confusionMatrix(table(predicted=as.numeric(levels(syn.y.pred)[as.integer(syn.y.pred
                                          actual = orig_data$target), mode = 'everything')
eval_mods <- data.frame(rf.org.cm$byClass,
                          rf.syn.cm$byClass)
eval_mods <- data.frame(t(eval_mods))
row.names(eval_mods) <- c("RF Original Tuned",
                          "RF Synthesized Tuned")
eval_mods <- dplyr::select(eval_mods, Sensitivity, Specificity, Precision, Recall, F1)

# ROC graph
ROCRPred.org <- prediction(as.numeric(levels(org.y.pred)[as.integer(org.y.pred)]),
                           orig_data$target)
ROCRPref.org <- performance(ROCRPred.org, 'tpr', 'fpr')

ROCRPred.syn <- prediction(as.numeric(levels(syn.y.pred)[as.integer(syn.y.pred)]),
                           orig_data$target)
ROCRPref.syn <- performance(ROCRPred.syn, 'tpr', 'fpr')
#-------------------------------------------------------------------------------------

# Hyper parameter tuning using Orig_data

## Using Caret Random Search
rf.random <- readRDS("./source/model/hp_rf_random.rds")
# rf.random <- train(target~.,
#                     data=orig_data,
#                     method="rf",
#                     metric="Accuracy",
#                     tuneLength=15,
#                     trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="random")
#                     preProcess = c("center", "scale"))
# saveRDS(rf.random, "./model/hp_rf_random.rds")
#-------------------------------------------------------------------------------------
## Grid Search
rf.gridsearch <- readRDS("./source/model/hp_rf_grid.rds")
# set.seed(seed)
# rf.gridsearch <- train(target~.,
#                         data=orig_data,
#                         method="rf",
#                         metric='Accuracy',
#                         tuneGrid=expand.grid(.mtry=c(1:15)),
#                         trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="grid
#                         preProcess = c("center", "scale"))
# saveRDS(rf.gridsearch, "./model/hp_rf_grid_1.rds")
#-------------------------------------------------------------------------------------
## Tuning using algorithm tools
bestmtry <- readRDS("./source/model/bestmtry.rds")
# bestmtry <- tuneRF(x, y, stepFactor=1.5, improve=1e-5, ntree=500)
# saveRDS(bestmtry, "./model/bestmtry.rds")
#-------------------------------------------------------------------------------------
## Craft your own - manually
manual.rf <- readRDS("./source/model/hp_manual.rds")
# modellist <- list()
# for (ntree in c(1000, 1500, 2000, 2500)) {
#   set.seed(seed)
#   fit <- train(target~., data=orig_data,
#                method="rf",
#                metric='Accuracy',
#                tuneGrid=expand.grid(.mtry=c(sqrt(ncol(x)))),
#                trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="grid"),
#                preProcess = c("center", "scale"),
#                ntree=ntree)
#   key <- toString(ntree)
#   modellist[[key]] <- fit
# }
# ### compare results
# manual.rf <- resamples(modellist)
# saveRDS(manual.rf, "./model/hp_manual.rds")
#-------------------------------------------------------------------------------------
## Extend Caret Custom
custom.result <- readRDS("./source/model/hp_custom.rds")
# custom.rf <- list(type = "Classification", library = "randomForest", loop = NULL)
#
# custom.rf$parameters <- data.frame(parameter = c("mtry", "ntree"),
#                                    class = rep("numeric", 2),
#                                    label = c("mtry", "ntree"))
#
# custom.rf$grid <- function(x, y, len = NULL, search = "grid") {}
#
# custom.rf$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
#   randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
# }
#
# custom.rf$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
#   predict(modelFit, newdata)
# custom.rf$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
#   predict(modelFit, newdata, type = "prob")
#
# custom.rf$sort <- function(x) x[order(x[,1]),]
# custom.rf$levels <- function(x) x$classes
#
# set.seed(seed)
# custom.result <- train(target~.,
#                        data=orig_data,
#                        method=custom.rf,
#                        metric='Accuracy',
#                        tuneGrid=expand.grid(.mtry=c(1:15), .ntree=c(1000, 1500, 2000, 2500)),
#                        trControl=trainControl(method="repeatedcv", number=10, repeats=3))
# saveRDS(custom.result, "./model/hp_custom.rds")
#-------------------------------------------------------------------------------------
# Hyper Parameter Tuning using syn_data <<<<<<<<<<<<<<<<<< TOO SLOW!

# ## Using Caret Random Search
# rf.random.syn <- train(target~.,
#                    data=syn_data,
#                    method="rf",
#                    metric="Accuracy",
#                    tuneLength=15,
#                    trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="random")
#                     preProcess = c("center", "scale"))
#
# ## Grid Search
# set.seed(seed)
# rf.gridsearch.syn <- train(target~.,
#                         data=syn_data,
#                         method="rf",
#                         metric=metric,
#                         tuneGrid=expand.grid(.mtry=c(1:15)),
#                         trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="grid
#                         preProcess = c("center", "scale"))
#
# ## Tuning using algorithm tools
# bestmtry.syn <- tuneRF(x, y, stepFactor=1.5, improve=1e-5, ntree=500)
#
# ## Craft your own - manually
# modellist.syn <- list()
# for (ntree in c(1000, 1500, 2000, 2500)) {
#   set.seed(seed)
#   fit <- train(target~., data=syn_data,
#                 method="rf",
#                 metric=metric,
#                 tuneGrid=expand.grid(.mtry=c(sqrt(ncol(x)))),
#                 trControl=trainControl(method="repeatedcv", number=10, repeats=3, search="grid"),
#                 preProcess = c("center", "scale"),
#                 ntree=ntree)
#   key <- toString(ntree)
#   modellist.syn[[key]] <- fit
# }
# ### compare results
# manual.rf.syn <- resamples(modellist.syn)
#
# ## Extend Caret Custom
# custom.rf.syn <- list(type = "Classification", library = "randomForest", loop = NULL)
#
# custom.rf.syn$parameters <- data.frame(parameter = c("mtry", "ntree"),
#                                    class = rep("numeric", 2),
#                                    label = c("mtry", "ntree"))
#
# custom.rf.syn$grid <- function(x, y, len = NULL, search = "grid") {}
#
# custom.rf.syn$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
#   randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
# }
#
# custom.rf.syn$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
#   predict(modelFit, newdata)
# custom.rf.syn$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
#   predict(modelFit, newdata, type = "prob")
#
# custom.rf.syn$sort <- function(x) x[order(x[,1]),]
# custom.rf.syn$levels <- function(x) x$classes
#
# set.seed(seed)
# custom.result.syn <- train(target~.,
#                         data=syn_data,
#                         method=custom.rf.syn,
#                         metric=metric,
#                         tuneGrid=expand.grid(.mtry=c(1:15), .ntree=c(1000, 1500, 2000, 2500)),
#                         trControl=trainControl(method="repeatedcv", number=10, repeats=3))

rf.org.cm2 <- caret::confusionMatrix(org.y.pred, reference=orig_data$target)
rf.syn.cm2 <- caret::confusionMatrix(syn.y.pred, reference=orig_data$target)

rf_compared <- data.frame(t(rf.org.cm2$overall))
rf_compared <- rbind(rf_compared, data.frame(t(rf.syn.cm2$overall)))

rownames(rf_compared) <- c("orig", "syn")
rf_compared <- rf_compared[order(-rf_compared$Accuracy),]
```

Table 6: Previous Approaches Documented by Shouman

AUTHOR	YEAR	TECHNIQUE	ACCURACY
Hall	2000	Na?ve Bayes	.832
Hall	2000	K Nearest Neighbour	.821
Hall	2000	Decision Tree	.753
Yan, Zheng et al.	2003	Multilayer Perceptron	.636
Herron	2004	Support Vector Machine	.836
Herron	2004	J4.8 Decision Tree	.776
Herron	2004	Support Vector Machine	.834
Andreeva	2006	Na?ve Bayes	.786
Andreeva	2006	Decision Tree	.757
Andreeva	2006	Neural Network	.828
Andreeva	2006	Sequential Minimal Optimization	.841
Andreeva	2006	Kernel Density	.844
Polat , Sahan et al.	2007	Fuzzy-AIRS-K-Nearest Neighbour	.870
Palaniappan and Awang	2007	Na?ve Bayes	.950
Palaniappan and Awang	2007	Decision Tree	.949
Palaniappan and Awang	2007	Neural Network	.935
De Beule, Maesa et al.	2007	Artificial Neural Network	.820
Tantimongcolwat, Naenna et al.	2008	Direct Kernel Self-organizing Map	.804
Tantimongcolwat, Naenna et al.	2008	Multilayer Perceptron	.745
Hara and Ichimura	2008	Automatically Defined Groups	.678
Hara and Ichimura	2008	Immune Multi-agent Neural Network	.823
Sitar-Taut, Zdrenghea et al.	2009	Na?ve Bayes	.620
Sitar-Taut, Zdrenghea et al.	2009	Decision Trees	.604
Tu, Shin et al.	2009	Bagging Algorithm	.814
Das, Turkoglu et al.	2009	Neural Network Ensembles	.890
Rajkumar and Reena	2010	Na?ve Bayes	.523
Rajkumar and Reena	2010	K Nearest Neighbour	.457
Rajkumar and Reena	2010	Decision List	.520
Srinivas, Rani et al.	2010	Na?ve Bayes	.841
Srinivas, Rani et al.	2010	One Dependency Augmented Na?ve Bayes	.805
Kangwanariyakul, Nantasenamat et al.	2010	Back-propagation Neural Network	.784
Kangwanariyakul, Nantasenamat et al.	2010	Bayesian Neural Network	.784
Kangwanariyakul, Nantasenamat et al.	2010	Probabilistic Neural Network	.706
Kangwanariyakul, Nantasenamat et al.	2010	Polynomial Support Vector Machine	.706
Kangwanariyakul, Nantasenamat et al.	2010	Radial Basis Support Vector Machine	.608
Kangwanariyakul, Nantasenamat et al.	2010	Bayesian Neural Network	.784
Kumari and Godara	2011	RIPPER	.811
Kumari and Godara	2011	Decision Tree	.791
Kumari and Godara	2011	Artificial Neural Network	,801
Kumari and Godara	2011	Support Vector Machine	.841
Soni, Ansari et al.	2011	Weighted Associative Classifier	.578
Soni, Ansari et al.	2011	Classification-Association	.583
Soni, Ansari et al.	2011	Classification-Multiple ClassAssociation	.536
Soni, Ansari et al.	2011	Classification-Predictive Association	.523
Abdullah and Rajalaxmi	2012	Decision Tree	.507
Abdullah and Rajalaxmi	2012	Random Forest	.633
Rajeswari, Vaithiyanathan et al.	2013	Neural Network	.805
Rajeswari, Vaithiyanathan et al.	2013	J4.8 Decision Tree	.779
Rajeswari, Vaithiyanathan et al.	2013	Support Vector Machine	.842
Rajeswari, Vaithiyanathan et al.	2013	Feature Selection with Neural Network	.845
Rajeswari, Vaithiyanathan et al.	2013	Feature Selection with Decision Tree	.842
Rajeswari, Vaithiyanathan et al.	2013	Feature Selection with Support Vector Machine	.875
Rajeswari, Vaithiyanathan et al.	2013	Neural Network	.805
Lakshmi, Krishna et al.	2013	Support Vector Machine	.781
Lakshmi, Krishna et al.	2013	Decision Tree	.847
Lakshmi, Krishna et al.	2013	K Nearest Neighbor	.840

Table 7: Average Performance by Technique

TECHNIQUE	MEDIAN ACCURACY
Logistic Regression	.855
Random Forest	.724
Support Vector Machine	.809
Naive Bayes	.819

Sensitivity	**Specificity**	**Precision**	**Recall**	**F1**
0.935	0.964	0.956	0.935	0.945
0.826	0.885	0.857	0.826	0.841